ZIP Code Geographic Analysis

Version 1.0

Generated by Doxygen 1.16.1

1 Class Index 2

1.1 Class List . . . 2

2 File Index 2

2.1 File List . . . 2

3 Class Documentation 2

3.1 StateExtremes Struct Reference . . . 2

3.1.1 Detailed Description . . . 3

3.1.2 Constructor & Destructor Documentation . . . 3

3.1.3 Member Data Documentation . . . 4

3.2 ZipCodeBuffer Class Reference . . . 6

3.2.1 Detailed Description . . . 7

3.2.2 Constructor & Destructor Documentation . . . 7

3.2.3 Member Function Documentation . . . 8

3.2.4 Member Data Documentation . . . 15

3.3 ZipCodeRecord Struct Reference . . . 16

3.3.1 Detailed Description . . . 17

3.3.2 Constructor & Destructor Documentation . . . 17

3.3.3 Member Data Documentation . . . 18

4 File Documentation 20

4.1 main.cpp File Reference . . . 20

4.1.1 Detailed Description . . . 20

4.1.2 USAGE . . . 21

4.1.3 OUTPUT . . . 21

4.1.4 Function Documentation . . . 21

4.2 main.cpp . . . 23

4.3 ZipCodeBuffer.cpp File Reference . . . 26

4.3.1 Detailed Description . . . 27

4.4 ZipCodeBuffer.cpp . . . 27

4.5 ZipCodeBuffer.h File Reference . . . 31

4.5.1 Detailed Description . . . 32

4.6 ZipCodeBuffer.h . . . 32

Index 37

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

StateExtremes
Holds the extreme ZIP codes for a single state 2

ZipCodeBuffer
A buffer class for reading ZIP code records from CSV files 6

ZipCodeRecord
Structure to hold a single ZIP code record 16

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

main.cpp
Main application program for ZIP code geographic analysis 20

ZipCodeBuffer.cpp
Implementation of the ZipCodeBuffer class 26

ZipCodeBuffer.h
Header file for the ZipCodeBuffer class 31

3 Class Documentation

3.1 StateExtremes Struct Reference

Holds the extreme ZIP codes for a single state.

Public Member Functions

- StateExtremes ()

  Constructor initializes all values to sentinel values.

Public Attributes

- int easternmost

    ZIP code with minimum longitude (farthest east).
- int westernmost

    ZIP code with maximum longitude (farthest west).
- int northernmost

    ZIP code with maximum latitude (farthest north).
- int southernmost

    ZIP code with minimum latitude (farthest south).
- double minLongitude

    Minimum longitude value (easternmost point).
- double maxLongitude

    Maximum longitude value (westernmost point).
- double maxLatitude

    Maximum latitude value (northernmost point).
- double minLatitude

    Minimum latitude value (southernmost point).

3.1.1 Detailed Description

Holds the extreme ZIP codes for a single state.

This structure stores the four extreme ZIP codes (by geographic coordinates) for a particular state. It is used to aggregate data during the analysis.

Definition at line 47 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

StateExtremes()

```
StateExtremes::StateExtremes () [inline]
```

Constructor initializes all values to sentinel values.

Uses extreme values so that any real coordinate will replace them during the first comparison.

Definition at line 64 of file main.cpp.

References easternmost, maxLatitude, maxLongitude, minLatitude, minLongitude, northernmost, southernmost, and westernmost.

3.1.3 Member Data Documentation

easternmost

```
int StateExtremes::easternmost
```

ZIP code with minimum longitude (farthest east).

Definition at line 48 of file main.cpp.

Referenced by calculateStateExtremes(), printStateExtremesTable(), and StateExtremes().

maxLatitude

```
double StateExtremes::maxLatitude
```

Maximum latitude value (northernmost point).

Definition at line 55 of file main.cpp.

Referenced by calculateStateExtremes(), and StateExtremes().

maxLongitude

```
double StateExtremes::maxLongitude
```

Maximum longitude value (westernmost point).

Definition at line 54 of file main.cpp.

Referenced by calculateStateExtremes(), and StateExtremes().

minLatitude

```
double StateExtremes::minLatitude
```

Minimum latitude value (southernmost point).

Definition at line 56 of file main.cpp.

Referenced by calculateStateExtremes(), and StateExtremes().

minLongitude

```
double StateExtremes::minLongitude
```

Minimum longitude value (easternmost point).

Definition at line 53 of file main.cpp.

Referenced by calculateStateExtremes(), and StateExtremes().

northernmost

```
int StateExtremes::northernmost
```

ZIP code with maximum latitude (farthest north).

Definition at line 50 of file main.cpp.

Referenced by calculateStateExtremes(), printStateExtremesTable(), and StateExtremes().

southernmost

```
int StateExtremes::southernmost
```

ZIP code with minimum latitude (farthest south).

Definition at line 51 of file main.cpp.

Referenced by calculateStateExtremes(), printStateExtremesTable(), and StateExtremes().

westernmost

```
int StateExtremes::westernmost
```

ZIP code with maximum longitude (farthest west).

Definition at line 49 of file main.cpp.

Referenced by calculateStateExtremes(), printStateExtremesTable(), and StateExtremes().

The documentation for this struct was generated from the following file:

- main.cpp

3.2 ZipCodeBuffer Class Reference

A buffer class for reading ZIP code records from CSV files.

```
#include <ZipCodeBuffer.h>
```

Public Member Functions

- ZipCodeBuffer ()

  Default constructor.
- ZipCodeBuffer (const string &csvFilename)

  Parameterized constructor.
- ∼ZipCodeBuffer ()

  Destructor.
- bool open (const string &csvFilename)

  Opens a CSV file for reading.
- void close ()

  Closes the currently open file.
- bool isOpen () const

  Checks if a file is currently open.
- bool readRecord (ZipCodeRecord &record)

  Reads the next ZIP code record from the file.
- vector< ZipCodeRecord > gatherAllRecords ()

  Reads and gathers all records from the file.
- bool reset ()

  Resets the file position to the beginning (after header).
- long getRecordCount () const

  Gets the total number of records read so far.
- string getFilename () const

  Gets the name of the currently open file.

Private Member Functions

- bool parseLine (const string &line, ZipCodeRecord &record)

  Parses a CSV line into a ZipCodeRecord.
- vector< string > splitCSV (const string &line)

  Splits a CSV line into individual fields.
- string trim (const string &str)

  Trims whitespace from both ends of a string.

Private Attributes

- ifstream fileStream

  Input file stream for reading CSV data.
- string filename

  Name of the CSV file being read.
- bool headerSkipped

  Flag to track if header row has been skipped.
- long recordCount

  Counter for total records read.

3.2.1 Detailed Description

A buffer class for reading ZIP code records from CSV files.

This class provides an abstraction layer for reading ZIP code data from a comma-separated values (CSV) file. It handles file I/O, parsing, and error checking while maintaining a clean interface for client code.

The class uses internal buffering to efficiently read data from the file and parse it into structured ZipCodeRecord objects.

Note

The CSV file must have a header row which is automatically skipped

Expected CSV format: ZipCode,PlaceName,State,County,Lat,Long

Definition at line 73 of file ZipCodeBuffer.h.

3.2.2 Constructor & Destructor Documentation

ZipCodeBuffer() [1/2]

```
ZipCodeBuffer::ZipCodeBuffer ()
```

Default constructor.

Creates an uninitialized ZipCodeBuffer. The open() method must be called before reading any records.

Default constructor implementation Initializes member variables to safe default values

Definition at line 51 of file ZipCodeBuffer.cpp.

References filename, headerSkipped, and recordCount.

ZipCodeBuffer() [2/2]

```
ZipCodeBuffer::ZipCodeBuffer (
            const string & csvFilename)  [explicit]
```

Parameterized constructor.

Parameters

csvFilename	Path to the CSV file to open

Creates a ZipCodeBuffer and automatically opens the specified file. The header row is skipped during initialization.

Parameterized constructor implementation Opens the specified file and prepares it for reading

Definition at line 59 of file ZipCodeBuffer.cpp.

References filename, headerSkipped, open(), and recordCount.

∼ZipCodeBuffer()

```
ZipCodeBuffer::∼ZipCodeBuffer ()
```

Destructor.

Ensures the file stream is properly closed when the object is destroyed, preventing resource leaks.

Destructor implementation Ensures file is properly closed to prevent resource leaks

Definition at line 68 of file ZipCodeBuffer.cpp.

References close().

3.2.3 Member Function Documentation

close()

```
void ZipCodeBuffer::close ()
```

Closes the currently open file.

Closes the file stream and resets internal state. Safe to call even if no file is open.

Closes the currently open file

Resets all internal state variables and closes the file stream. Safe to call multiple times or when no file is open.

Definition at line 121 of file ZipCodeBuffer.cpp.

References filename, fileStream, headerSkipped, and recordCount.

Referenced by main(), open(), and ∼ZipCodeBuffer().

gatherAllRecords()

```
vector< ZipCodeRecord > ZipCodeBuffer::gatherAllRecords ()
```

Reads and gathers all records from the file.

Returns

> Vector containing all ZIP code records from the file

Reads the entire CSV file and returns all valid records as a vector. The file position is reset to the beginning (after header) when complete.

Note

> This method loads all data into memory - use with caution for very large files

Reads all records from the file into a vector

This method reads the entire file and returns all valid records. The file position is reset to the beginning after reading.

Returns

> Vector containing all ZipCodeRecord objects from the file

Definition at line 183 of file ZipCodeBuffer.cpp.

References readRecord(), and reset().

Referenced by main().

getFilename()

```
string ZipCodeBuffer::getFilename () const
```

Gets the name of the currently open file.

Returns

Filename string

Gets the filename of the currently open file

Returns

String containing the filename

Definition at line 245 of file ZipCodeBuffer.cpp.

References filename.

getRecordCount()

```
long ZipCodeBuffer::getRecordCount () const
```

Gets the total number of records read so far.

Returns

Count of records read

Gets the count of records read so far

Returns

Number of records successfully read

Definition at line 236 of file ZipCodeBuffer.cpp.

References recordCount.

isOpen()

```
bool ZipCodeBuffer::isOpen () const
```

Checks if a file is currently open.

Returns

true if file is open and ready for reading, false otherwise

Checks if file is open and ready for reading

Returns

true if file stream is open, false otherwise

Definition at line 135 of file ZipCodeBuffer.cpp.

References fileStream.

open()

```
bool ZipCodeBuffer::open (
             const string & csvFilename)
```

Opens a CSV file for reading.

Parameters

csvFilename	Path to the CSV file

Returns

true if file opened successfully, false otherwise

Opens the specified CSV file and skips the header row. If a file is already open, it is closed first.

Opens a CSV file for reading

This method:

1. Closes any previously open file
2. Opens the new file in input mode
3. Skips the header row
4. Resets the record counter

Parameters

csvFilename	Path to the CSV file to open

Returns

true if successful, false if file cannot be opened

Definition at line 84 of file ZipCodeBuffer.cpp.

References close(), filename, fileStream, headerSkipped, and recordCount.

Referenced by main(), and ZipCodeBuffer().

parseLine()

```
bool ZipCodeBuffer::parseLine (
            const string & line,
            ZipCodeRecord & record)  [private]
```

Parses a CSV line into a ZipCodeRecord.

Parameters

line	The CSV line to parse
record	Reference to ZipCodeRecord to populate

Returns

true if parsing was successful, false otherwise

This private helper method takes a raw CSV line and extracts the individual fields, converting them to appropriate data types and storing them in the provided ZipCodeRecord structure.

Handles quoted fields and embedded commas correctly.

Parses a CSV line into a ZipCodeRecord

This private helper method handles the conversion of string fields to appropriate data types and performs basic validation.

Expected CSV format: ZipCode,PlaceName,State,County,Lat,Long

Parameters

line	The CSV line to parse

record	Reference to ZipCodeRecord to populate

Returns

true if parsing succeeded, false if format is invalid

Definition at line 261 of file ZipCodeBuffer.cpp.

References ZipCodeRecord::county, ZipCodeRecord::latitude, ZipCodeRecord::longitude, ZipCodeRecord::placeName, splitCSV(), ZipCodeRecord::state, trim(), and ZipCodeRecord::zipCode.

Referenced by readRecord().

readRecord()

```
bool ZipCodeBuffer::readRecord (
            ZipCodeRecord & record)
```

Reads the next ZIP code record from the file.

Parameters

record	Reference to ZipCodeRecord to populate

Returns

true if a record was successfully read, false on EOF or error

Reads one line from the CSV file, parses it, and populates the provided ZipCodeRecord structure. Returns false when end of file is reached or if a parsing error occurs.

Note

Automatically skips the header row on first read

Reads a single record from the CSV file

This method reads one line from the file, parses it into fields, and populates the provided ZipCodeRecord structure.

Parameters

record	Reference to ZipCodeRecord to be populated

Returns

true if a record was read successfully, false on EOF or error

Definition at line 148 of file ZipCodeBuffer.cpp.

References fileStream, parseLine(), readRecord(), recordCount, and trim().

Referenced by gatherAllRecords(), and readRecord().

reset()

```
bool ZipCodeBuffer::reset ()
```

Resets the file position to the beginning (after header).

Returns

true if reset was successful, false otherwise

Seeks back to the start of the file and skips the header row again, allowing the file to be re-read without closing and reopening.

Resets the file position to the beginning (after header)

This allows the file to be re-read without closing and reopening.

Returns

true if reset was successful, false otherwise

Definition at line 208 of file ZipCodeBuffer.cpp.

References fileStream, and recordCount.

Referenced by gatherAllRecords().

splitCSV()

```
vector< string > ZipCodeBuffer::splitCSV (
            const string & line) [private]
```

Splits a CSV line into individual fields.

Parameters

line	The CSV line to split

Returns

Vector of strings containing individual fields

This utility function correctly handles CSV formatting including:

- Quoted fields
- Embedded commas within quotes
- Leading/trailing whitespace

Splits a CSV line into individual fields

This method correctly handles:

- Regular comma-separated fields
- Quoted fields containing commas
- Embedded quotes (escaped as "")

Algorithm:

1. Iterate through each character
2. Track whether we're inside quotes
3. Split on commas that are not inside quotes

Parameters

line	The CSV line to split

Returns

Vector of field strings

Definition at line 302 of file ZipCodeBuffer.cpp.

Referenced by parseLine().

trim()

```
string ZipCodeBuffer::trim (
             const string & str)  [private]
```

Trims whitespace from both ends of a string.

Parameters

str	The string to trim

Returns

Trimmed string

Trims leading and trailing whitespace from a string

This utility function removes spaces, tabs, newlines, and other whitespace characters from both ends of the string.

Parameters

str	The string to trim

Returns

Trimmed string

Definition at line 338 of file ZipCodeBuffer.cpp.

Referenced by parseLine(), and readRecord().

3.2.4 Member Data Documentation

filename

```
string ZipCodeBuffer::filename  [private]
```

Name of the CSV file being read.

Definition at line 76 of file ZipCodeBuffer.h.

Referenced by close(), getFilename(), open(), ZipCodeBuffer(), and ZipCodeBuffer().

fileStream

```
ifstream ZipCodeBuffer::fileStream  [private]
```

Input file stream for reading CSV data.

Definition at line 75 of file ZipCodeBuffer.h.

Referenced by close(), isOpen(), open(), readRecord(), and reset().

headerSkipped

```
bool ZipCodeBuffer::headerSkipped  [private]
```

Flag to track if header row has been skipped.

Definition at line 77 of file ZipCodeBuffer.h.

Referenced by close(), open(), ZipCodeBuffer(), and ZipCodeBuffer().

recordCount

```
long ZipCodeBuffer::recordCount  [private]
```

Counter for total records read.

Definition at line 78 of file ZipCodeBuffer.h.

Referenced by close(), getRecordCount(), open(), readRecord(), reset(), ZipCodeBuffer(), and ZipCodeBuffer().

The documentation for this class was generated from the following files:

- ZipCodeBuffer.h
- ZipCodeBuffer.cpp

3.3 ZipCodeRecord Struct Reference

Structure to hold a single ZIP code record.

```
#include <ZipCodeBuffer.h>
```

Public Member Functions

- ZipCodeRecord ()

    Default constructor.
- ZipCodeRecord (int zip, const string &place, const string &st, const string &cnty, double lat, double lon)

    Parameterized constructor.

Public Attributes

- int zipCode

  The 5-digit ZIP code.
- string placeName

  Name of the place/city.
- string state

  Two-letter state abbreviation.
- string county

  County name.
- double latitude

  Latitude coordinate (decimal degrees).
- double longitude

  Longitude coordinate (decimal degrees).

3.3.1 Detailed Description

Structure to hold a single ZIP code record.

This structure represents one row from the ZIP code CSV file, containing all relevant geographic and administrative information for a specific ZIP code.

Definition at line 30 of file ZipCodeBuffer.h.

3.3.2 Constructor & Destructor Documentation

ZipCodeRecord() `[1/2]`

```
ZipCodeRecord::ZipCodeRecord ()
```

Default constructor.

Initializes all numeric fields to zero and strings to empty.

Default constructor implementation Initializes all fields to default values

Definition at line 28 of file ZipCodeBuffer.cpp.

References county, latitude, longitude, placeName, state, and zipCode.

ZipCodeRecord() [2/2]

```
ZipCodeRecord::ZipCodeRecord (
             int zip,
             const string & place,
             const string & st,
             const string & cnty,
             double lat,
             double lon)
```

Parameterized constructor.

Parameters

zip	ZIP code number
place	Place name
st	State abbreviation
cnty	County name
lat	Latitude
lon	Longitude

Parameterized constructor implementation Initializes all fields with provided values

Definition at line 36 of file ZipCodeBuffer.cpp.

References county, latitude, longitude, placeName, state, and zipCode.

3.3.3 Member Data Documentation

county

```
string ZipCodeRecord::county
```

County name.

Definition at line 34 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

latitude

```
double ZipCodeRecord::latitude
```

Latitude coordinate (decimal degrees).

Definition at line 35 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

longitude

```
double ZipCodeRecord::longitude
```

Longitude coordinate (decimal degrees).

Definition at line 36 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

placeName

```
string ZipCodeRecord::placeName
```

Name of the place/city.

Definition at line 32 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

state

```
string ZipCodeRecord::state
```

Two-letter state abbreviation.

Definition at line 33 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

zipCode

```
int ZipCodeRecord::zipCode
```

The 5-digit ZIP code.

Definition at line 31 of file ZipCodeBuffer.h.

Referenced by ZipCodeBuffer::parseLine(), ZipCodeRecord(), and ZipCodeRecord().

The documentation for this struct was generated from the following files:

- ZipCodeBuffer.h
- ZipCodeBuffer.cpp

4 File Documentation

4.1 main.cpp File Reference

Main application program for ZIP code geographic analysis.

```
#include "ZipCodeBuffer.h"
#include <iostream>
#include <map>
#include <iomanip>
#include <algorithm>
#include <limits>
```

Classes

- struct StateExtremes

  Holds the extreme ZIP codes for a single state.

Functions

- map< string, StateExtremes > calculateStateExtremes (const vector< ZipCodeRecord > &records)

  Processes all ZIP code records and determines state extremes.
- void printStateExtremesTable (const map< string, StateExtremes > &stateMap)

  Prints a formatted table of state extremes to stdout.
- int main (int argc, char ∗argv[ ])

  Main program entry point.

4.1.1 Detailed Description

Main application program for ZIP code geographic analysis.

Author

Teagen Lee, ADD NAMES

Date

February 2026

This program analyzes ZIP code data from a CSV file and generates a report showing the extreme geographic coordinates (Easternmost, Westernmost, Northernmost, and Southernmost ZIP codes) for each state.

The program demonstrates:

- Use of the ZipCodeBuffer class for CSV parsing
- Data aggregation and analysis
- Formatted console output
- Proper resource management

4.1.2 USAGE

Usage: ./zip_analysis <csv_filename>

4.1.3 OUTPUT

The program outputs a formatted table to stdout with the following columns:

- State: Two-letter state abbreviation
- Easternmost: ZIP code with least (most negative) longitude
- Westernmost: ZIP code with greatest (most positive) longitude

- Northernmost: ZIP code with greatest (most positive) latitude
- Southernmost: ZIP code with least (most negative) latitude

Definition in file main.cpp.

4.1.4 Function Documentation

calculateStateExtremes()

```
map< string, StateExtremes > calculateStateExtremes (
             const vector< ZipCodeRecord > & records)
```

Processes all ZIP code records and determines state extremes.

Parameters

records	Vector of all ZIP code records

Returns

Map of state abbreviations to their StateExtremes data

This function iterates through all records and maintains running extremes for each state. For each record:

- If the longitude is less than current minimum, update easternmost
- If the longitude is greater than current maximum, update westernmost
- If the latitude is greater than current maximum, update northernmost
- If the latitude is less than current minimum, update southernmost

Note

Longitude in the US is negative (west of Prime Meridian) So MINIMUM longitude = EASTERNMOST point And MAXIMUM longitude = WESTERNMOST point

Latitude is positive in Northern Hemisphere So MAXIMUM latitude = NORTHERNMOST point And MINIMUM latitude = SOUTHERNMOST point

Definition at line 92 of file main.cpp.

References StateExtremes::easternmost, StateExtremes::maxLatitude, StateExtremes::maxLongitude, StateExtremes::minLatitude, StateExtremes::minLongitude, StateExtremes::northernmost, StateExtremes::southernmost, and StateExtremes::westernmost.

Referenced by main().

main()

```
int main (
            int argc,
            char * argv[])
```

Main program entry point.

Parameters

argc	Number of command line arguments
argv	Array of command line argument strings

Returns

0 on success, non-zero on error

Program flow:

1. Validate command line arguments
2. Open CSV file using ZipCodeBuffer
3. Read all records into memory
4. Calculate extreme coordinates for each state
5. Display formatted results
6. Clean up and exit

Definition at line 213 of file main.cpp.

References calculateStateExtremes(), ZipCodeBuffer::close(), ZipCodeBuffer::gatherAllRecords(), ZipCodeBuffer::open(), and printStateExtremesTable().

printStateExtremesTable()

```
void printStateExtremesTable (
            const map< string, StateExtremes > & stateMap)
```

Prints a formatted table of state extremes to stdout.

Parameters

stateMap	Map containing StateExtremes for each state

This function generates a nicely formatted table with:

- A header row with column labels
- One row per state, alphabetically sorted
- Aligned columns for readability
- ZIP codes formatted as 5-digit numbers

The output format is designed to be clear and professional, suitable for reports or further processing.

Definition at line 155 of file main.cpp.

References StateExtremes::easternmost, StateExtremes::northernmost, StateExtremes::southernmost, and StateExtremes::westernmost

Referenced by main().

4.2 main.cpp

Go to the documentation of this file.

```
/**
 * @file main.cpp
 * @brief Main application program for ZIP code geographic analysis
 * @author Teagen Lee, ADD NAMES
 * @date February 2026
 *
 * This program analyzes ZIP code data from a CSV file and generates a report
 * showing the extreme geographic coordinates (Easternmost, Westernmost,
 * Northernmost, and Southernmost ZIP codes) for each state.
 *
 * The program demonstrates:
 * - Use of the ZipCodeBuffer class for CSV parsing
 * - Data aggregation and analysis
 * - Formatted console output
 * - Proper resource management
 *
 * @section USAGE
 * Usage: ./zip_analysis <csv_filename>
 *
 * @section OUTPUT
 * The program outputs a formatted table to stdout with the following columns:
 * - State: Two-letter state abbreviation
 * - Easternmost: ZIP code with least (most negative) longitude
 * - Westernmost: ZIP code with greatest (most positive) longitude
 * - Northernmost: ZIP code with greatest (most positive) latitude
 * - Southernmost: ZIP code with least (most negative) latitude
 */

#include "ZipCodeBuffer.h"
```

```cpp
#include <iostream>
#include <map>
#include <iomanip>
#include <algorithm>
#include <limits>

#include "ZipCodeBuffer.h"

using namespace std;

/**
 * @struct StateExtremes
 * @brief Holds the extreme ZIP codes for a single state
 *
 * This structure stores the four extreme ZIP codes (by geographic coordinates)
 * for a particular state. It is used to aggregate data during the analysis.
 */
struct StateExtremes {
    int easternmost;     ///< ZIP code with minimum longitude (farthest east)
    int westernmost;     ///< ZIP code with maximum longitude (farthest west)
    int northernmost;    ///< ZIP code with maximum latitude (farthest north)
    int southernmost;    ///< ZIP code with minimum latitude (farthest south)

    double minLongitude; ///< Minimum longitude value (easternmost point)
    double maxLongitude; ///< Maximum longitude value (westernmost point)
    double maxLatitude;  ///< Maximum latitude value (northernmost point)
    double minLatitude;  ///< Minimum latitude value (southernmost point)

    /**
     * @brief Constructor initializes all values to sentinel values
     *
     * Uses extreme values so that any real coordinate will replace them
     * during the first comparison.
     */
    StateExtremes()
        : easternmost(0), westernmost(0), northernmost(0), southernmost(0),
          minLongitude(numeric_limits<double>::max()),
          maxLongitude(numeric_limits<double>::lowest()),
          maxLatitude(numeric_limits<double>::lowest()),
          minLatitude(numeric_limits<double>::max()) {
    }
};

/**
 * @brief Processes all ZIP code records and determines state extremes
 * @param records Vector of all ZIP code records
 * @return Map of state abbreviations to their StateExtremes data
 *
 * This function iterates through all records and maintains running
 * extremes for each state. For each record:
 * - If the longitude is less than current minimum, update easternmost
 * - If the longitude is greater than current maximum, update westernmost
 * - If the latitude is greater than current maximum, update northernmost
 * - If the latitude is less than current minimum, update southernmost
 *
 * @note Longitude in the US is negative (west of Prime Meridian)
 *       So MINIMUM longitude = EASTERNMOST point
 *       And MAXIMUM longitude = WESTERNMOST point
 * @note Latitude is positive in Northern Hemisphere
 *       So MAXIMUM latitude = NORTHERNMOST point
 *       And MINIMUM latitude = SOUTHERNMOST point
 */
map<string, StateExtremes> calculateStateExtremes(
    const vector<ZipCodeRecord>& records) {

    map<string, StateExtremes> stateMap;

    // Iterate through all records
    for (const auto& record : records) {
        const string& state = record.state;
        StateExtremes& extremes = stateMap[state]; // Creates entry if doesn’t exist

        /*
         * Check and update EASTERNMOST (minimum longitude)
         * Longitude in US is negative, so more negative = farther east
         */
        if (record.longitude < extremes.minLongitude) {
            extremes.minLongitude = record.longitude;
            extremes.easternmost = record.zipCode;
        }

        /*
         * Check and update WESTERNMOST (maximum longitude)
         * Less negative (closer to 0) = farther west
         */
        if (record.longitude > extremes.maxLongitude) {
            extremes.maxLongitude = record.longitude;
            extremes.westernmost = record.zipCode;
        }

        /*
         * Check and update NORTHERNMOST (maximum latitude)
         * Higher latitude = farther north
         */
        if (record.latitude > extremes.maxLatitude) {
            extremes.maxLatitude = record.latitude;
            extremes.northernmost = record.zipCode;
        }

        /*
         * Check and update SOUTHERNMOST (minimum latitude)
         * Lower latitude = farther south
         */
        if (record.latitude < extremes.minLatitude) {
            extremes.minLatitude = record.latitude;
            extremes.southernmost = record.zipCode;
        }
    }

    return stateMap;
}

/**
 * @brief Prints a formatted table of state extremes to stdout
 * @param stateMap Map containing StateExtremes for each state
 *
 * This function generates a nicely formatted table with:
 * - A header row with column labels
 * - One row per state, alphabetically sorted
 * - Aligned columns for readability
 * - ZIP codes formatted as 5-digit numbers
 *
 * The output format is designed to be clear and professional,
 * suitable for reports or further processing.
 */
void printStateExtremesTable(const map<string, StateExtremes>& stateMap) {
    // Print header row with column labels
    cout « left;  // Left-align text
         cout « setw(8) « "State"
              « setw(15) « "Easternmost"
              « setw(15) « "Westernmost"
              « setw(15) « "Northernmost"
              « setw(15) « "Southernmost"
              « endl;

    // Print separator line for visual clarity
    cout « string(68, '-') « endl;

    /*
     * Print data rows
     * The map automatically keeps states in alphabetical order by key
     * because map maintains sorted order
     */
    for (const auto& entry : stateMap) {
        const string& state = entry.first;
        const StateExtremes& extremes = entry.second;

        // Print state abbreviation and ZIP codes
        // Format each ZIP code separately with proper padding
        cout « setw(8) « state;

        // Easternmost ZIP code
        cout « setfill('0') « setw(5) « extremes.easternmost
                  « setfill(' ') « setw(10) « " ";

        // Westernmost ZIP code
        cout « setfill('0') « setw(5) « extremes.westernmost
                  « setfill(' ') « setw(10) « " ";

        // Northernmost ZIP code
        cout « setfill('0') « setw(5) « extremes.northernmost
                  « setfill(' ') « setw(10) « " ";

        // Southernmost ZIP code
        cout « setfill('0') « setw(5) « extremes.southernmost
             « setfill(' ') « endl;
    }
}

/**
 * @brief Main program entry point
 * @param argc Number of command line arguments
 * @param argv Array of command line argument strings
 * @return 0 on success, non-zero on error
 *
 * Program flow:
 * 1. Validate command line arguments
 * 2. Open CSV file using ZipCodeBuffer
 * 3. Read all records into memory
 * 4. Calculate extreme coordinates for each state
 * 5. Display formatted results
 * 6. Clean up and exit
 */
int main(int argc, char* argv[]) {
    // Check command line arguments
    if (argc != 2) {
        cerr « "Usage: " « argv[0] « " <csv_filename>" « endl;
        cerr « "Example: " « argv[0] « " us_postal_codes.csv" « endl;
        return 1;
    }

    // Get filename from command line
    string filename = argv[1];

    // Create ZipCodeBuffer object
    ZipCodeBuffer buffer;

    // Attempt to open the CSV file
    if (!buffer.open(filename)) {
        cerr « "Error: Could not open file '" « filename « "'" « endl;
        cerr « "Please check that the file exists and is readable." « endl;
        return 2;
    }

    cout « "Reading ZIP code data from: " « filename « endl;
    cout « "Processing records..." « endl « endl;

    // Read all records from the file
    vector<ZipCodeRecord> allRecords = buffer.gatherAllRecords();

    // Check if we got any records
    if (allRecords.empty()) {
        cerr « "Error: No valid records found in file." « endl;
        buffer.close();
        return 3;
    }

    cout « "Total records read: " « allRecords.size() « endl « endl;

    // Calculate state extremes
    map<string, StateExtremes> stateExtremes = calculateStateExtremes(allRecords);

    cout « "Analysis Results:" « endl;
    cout « "=================" « endl « endl;

    // Print the results table
    printStateExtremesTable(stateExtremes);

    cout « endl;
    cout « "Total states/territories: " « stateExtremes.size() « endl;

    // Close the file (destructor would do this automatically, but being explicit)
    buffer.close();

    return 0; // Success
}
```

4.3 ZipCodeBuffer.cpp File Reference

Implementation of the ZipCodeBuffer class.

```
#include "ZipCodeBuffer.h"
#include <algorithm>
#include <cctype>
```

4.3.1 Detailed Description

Implementation of the ZipCodeBuffer class.

Author

Teagen Lee, ADD NAMES

Date

February 2026

This file contains the implementation of all methods declared in ZipCodeBuffer.h. The class provides robust CSV parsing with proper error handling and memory management.

Definition in file ZipCodeBuffer.cpp.

4.4 ZipCodeBuffer.cpp

Go to the documentation of this file.

```
/**
 * @file ZipCodeBuffer.cpp
 * @brief Implementation of the ZipCodeBuffer class
 * @author Teagen Lee, ADD NAMES
 * @date February 2026
 *
 * This file contains the implementation of all methods declared in
 * ZipCodeBuffer.h. The class provides robust CSV parsing with proper
 * error handling and memory management.
 */

#include "ZipCodeBuffer.h"
#include <algorithm>
#include <cctype>

using namespace std;

/*
 * ============================================================================
 * ZipCodeRecord Implementation
 * ============================================================================
 */

/**
 * Default constructor implementation
 * Initializes all fields to default values
 */
ZipCodeRecord::ZipCodeRecord()
    : zipCode(0), placeName(""), state(""), county(""), latitude(0.0), longitude(0.0) {
}

/**
 * Parameterized constructor implementation
 * Initializes all fields with provided values
 */
ZipCodeRecord::ZipCodeRecord(int zip, const string& place, const string& st,
                               const string& cnty, double lat, double lon)
    : zipCode(zip), placeName(place), state(st), county(cnty), latitude(lat), longitude(lon) {
}

/*
 * ============================================================================
 * ZipCodeBuffer Implementation
 * ============================================================================
 */

/**
 * Default constructor implementation
 * Initializes member variables to safe default values
 */
ZipCodeBuffer::ZipCodeBuffer()
    : filename(""), headerSkipped(false), recordCount(0) {
}

/**
 * Parameterized constructor implementation
 * Opens the specified file and prepares it for reading
 */
ZipCodeBuffer::ZipCodeBuffer(const string& csvFilename)
    : filename(""), headerSkipped(false), recordCount(0) {
    open(csvFilename);
}

/**
 * Destructor implementation
 * Ensures file is properly closed to prevent resource leaks
 */
ZipCodeBuffer::~ZipCodeBuffer() {
    close();
}

/**
 * Opens a CSV file for reading
 *
 * This method:
 * 1. Closes any previously open file
 * 2. Opens the new file in input mode
 * 3. Skips the header row
 * 4. Resets the record counter
 *
 * @param csvFilename Path to the CSV file to open
 * @return true if successful, false if file cannot be opened
 */
bool ZipCodeBuffer::open(const string& csvFilename) {
    // Close any currently open file
    close();

    // Store the filename
    filename = csvFilename;

    // Open the file
    fileStream.open(filename);

    // Check if file opened successfully
    if (!fileStream.is_open()) {
        return false;
    }

    // Skip the header row
    string headerLine;
    if (getline(fileStream, headerLine)) {
        headerSkipped = true;
    } else {
        // File is empty or unreadable
        close();
        return false;
    }

    // Reset record counter
    recordCount = 0;

    return true;
}

/**
 * Closes the currently open file
 *
 * Resets all internal state variables and closes the file stream.
 * Safe to call multiple times or when no file is open.
 */
void ZipCodeBuffer::close() {
    if (fileStream.is_open()) {
        fileStream.close();
    }
    filename = "";
    headerSkipped = false;
    recordCount = 0;
}

/**
 * Checks if file is open and ready for reading
 *
 * @return true if file stream is open, false otherwise
 */
bool ZipCodeBuffer::isOpen() const {
    return fileStream.is_open();
}

/**
 * Reads a single record from the CSV file
 *
 * This method reads one line from the file, parses it into fields,
 * and populates the provided ZipCodeRecord structure.
 *
 * @param record Reference to ZipCodeRecord to be populated
 * @return true if a record was read successfully, false on EOF or error
 */
bool ZipCodeBuffer::readRecord(ZipCodeRecord& record) {
    // Check if file is open
    if (!fileStream.is_open()) {
        return false;
    }

    // Read a line from the file
    string line;
    if (getline(fileStream, line)) {
        // Skip empty lines
        if (line.empty() || trim(line).empty()) {
            return readRecord(record); // Recursively read next non-empty line
        }

        // Parse the line into a record
        if (parseLine(line, record)) {
            recordCount++;
            return true;
        } else {
            return false; // Parse error
        }
    }

    // End of file reached
    return false;
}

/**
 * Reads all records from the file into a vector
 *
 * This method reads the entire file and returns all valid records.
 * The file position is reset to the beginning after reading.
 *
 * @return Vector containing all ZipCodeRecord objects from the file
 */
vector<ZipCodeRecord> ZipCodeBuffer::gatherAllRecords() {
    vector<ZipCodeRecord> records;

    // Reset to beginning of file (after header)
    reset();

    // Read all records
    ZipCodeRecord record;
    while (readRecord(record)) {
        records.push_back(record);
    }

    // Reset again so file can be re-read if needed
    reset();

    return records;
```

```
}

/**
 * Resets the file position to the beginning (after header)
 *
 * This allows the file to be re-read without closing and reopening.
 *
 * @return true if reset was successful, false otherwise
 */
bool ZipCodeBuffer::reset() {
    if (!fileStream.is_open()) {
        return false;
    }

    // Clear any error flags
    fileStream.clear();

    // Seek to beginning of file
    fileStream.seekg(0, ios::beg);

    // Skip header row again
    string headerLine;
    if (!getline(fileStream, headerLine)) {
        return false;
    }

    // Reset record counter
    recordCount = 0;

    return true;
}

/**
 * Gets the count of records read so far
 *
 * @return Number of records successfully read
 */
long ZipCodeBuffer::getRecordCount() const {
    return recordCount;
}

/**
 * Gets the filename of the currently open file
 *
 * @return String containing the filename
 */
string ZipCodeBuffer::getFilename() const {
    return filename;
}

/**
 * Parses a CSV line into a ZipCodeRecord
 *
 * This private helper method handles the conversion of string fields
 * to appropriate data types and performs basic validation.
 *
 * Expected CSV format: ZipCode,PlaceName,State,County,Lat,Long
 *
 * @param line The CSV line to parse
 * @param record Reference to ZipCodeRecord to populate
 * @return true if parsing succeeded, false if format is invalid
 */
bool ZipCodeBuffer::parseLine(const string& line, ZipCodeRecord& record) {
    // Split the line into fields
    vector<string> fields = splitCSV(line);

    // Verify we have the correct number of fields
    if (fields.size() != 6) {
        return false;
    }

    try {
        // Parse each field with appropriate type conversion
        record.zipCode = stoi(trim(fields[0]));
        record.placeName = trim(fields[1]);
        record.state = trim(fields[2]);
        record.county = trim(fields[3]);
        record.latitude = stod(trim(fields[4]));
        record.longitude = stod(trim(fields[5]));

        return true;
    } catch (const exception& e) {
        // Conversion failed - invalid data format
        return false;
    }
}

/**
 * Splits a CSV line into individual fields
 *
 * This method correctly handles:
 * - Regular comma-separated fields
 * - Quoted fields containing commas
 * - Embedded quotes (escaped as "")
 *
 * Algorithm:
 * 1. Iterate through each character
 * 2. Track whether we're inside quotes
 * 3. Split on commas that are not inside quotes
 *
 * @param line The CSV line to split
 * @return Vector of field strings
 */
vector<string> ZipCodeBuffer::splitCSV(const string& line) {
    vector<string> fields;
    string currentField;
    bool inQuotes = false;

    for (size_t i = 0; i < line.length(); i++) {
        char c = line[i];

        if (c == '"') {
            // Toggle quote state
            inQuotes = !inQuotes;
        } else if (c == ',' && !inQuotes) {
            // Field separator found (not inside quotes)
            fields.push_back(currentField);
            currentField.clear();
        } else {
            // Regular character - add to current field
            currentField += c;
        }
    }

    // Add the last field
    fields.push_back(currentField);

    return fields;
}

/**
 * Trims leading and trailing whitespace from a string
 *
 * This utility function removes spaces, tabs, newlines, and other
 * whitespace characters from both ends of the string.
 *
 * @param str The string to trim
 * @return Trimmed string
 */
string ZipCodeBuffer::trim(const string& str) {
    // Find first non-whitespace character
    size_t start = str.find_first_not_of(" \t\r\n");
    if (start == string::npos) {
        return ""; // String is all whitespace
    }

    // Find last non-whitespace character
    size_t end = str.find_last_not_of(" \t\r\n");

    // Extract substring
    return str.substr(start, end - start + 1);
}
```

4.5 ZipCodeBuffer.h File Reference

Header file for the ZipCodeBuffer class.

```
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include "ZipCodeBuffer.cpp"
```

Classes

- struct ZipCodeRecord

  Structure to hold a single ZIP code record.
- class ZipCodeBuffer

  A buffer class for reading ZIP code records from CSV files.

4.5.1 Detailed Description

Header file for the ZipCodeBuffer class.

Author

Teagen Lee, ADD NAMES

Date

February 2026

This file contains the declaration of the ZipCodeBuffer class which provides functionality to read and parse ZIP code records from a CSV file. The class implements buffered reading for efficient file I/O operations.

Definition in file ZipCodeBuffer.h.

4.6 ZipCodeBuffer.h

Go to the documentation of this file.

```
/**
 * @file ZipCodeBuffer.h
 * @brief Header file for the ZipCodeBuffer class
 * @author Teagen Lee, ADD NAMES
 * @date February 2026
 *
 * This file contains the declaration of the ZipCodeBuffer class which provides
 * functionality to read and parse ZIP code records from a CSV file. The class
 * implements buffered reading for efficient file I/O operations.
 */

#ifndef ZIPCODEBUFFER_H
#define ZIPCODEBUFFER_H

#include <string>
#include <fstream>
#include <sstream>
#include <vector>

using namespace std;
```

```

/**
 * @struct ZipCodeRecord
 * @brief Structure to hold a single ZIP code record
 *
 * This structure represents one row from the ZIP code CSV file,
 * containing all relevant geographic and administrative information
 * for a specific ZIP code.
 */
struct ZipCodeRecord {
    int zipCode;           ///< The 5-digit ZIP code
    string placeName; ///< Name of the place/city
    string state;     ///< Two-letter state abbreviation
    string county;    ///< County name
    double latitude;       ///< Latitude coordinate (decimal degrees)
    double longitude;      ///< Longitude coordinate (decimal degrees)

    /**
     * @brief Default constructor
     *
     * Initializes all numeric fields to zero and strings to empty.
     */
    ZipCodeRecord();

    /**
     * @brief Parameterized constructor
     * @param zip ZIP code number
     * @param place Place name
     * @param st State abbreviation
     * @param cnty County name
     * @param lat Latitude
     * @param lon Longitude
     */
    ZipCodeRecord(int zip, const string& place, const string& st,
                  const string& cnty, double lat, double lon);
};

/**
 * @class ZipCodeBuffer
 * @brief A buffer class for reading ZIP code records from CSV files
 *
 * This class provides an abstraction layer for reading ZIP code data
 * from a comma-separated values (CSV) file. It handles file I/O,
 * parsing, and error checking while maintaining a clean interface
 * for client code.
 *
 * The class uses internal buffering to efficiently read data from
 * the file and parse it into structured ZipCodeRecord objects.
 *
 * @note The CSV file must have a header row which is automatically skipped
 * @note Expected CSV format: ZipCode,PlaceName,State,County,Lat,Long
 */
class ZipCodeBuffer {
private:
    ifstream fileStream;  ///< Input file stream for reading CSV data
    string filename;      ///< Name of the CSV file being read
    bool headerSkipped;        ///< Flag to track if header row has been skipped
    long recordCount;          ///< Counter for total records read

    /**
     * @brief Parses a CSV line into a ZipCodeRecord
     * @param line The CSV line to parse
     * @param record Reference to ZipCodeRecord to populate
     * @return true if parsing was successful, false otherwise
     *
     * This private helper method takes a raw CSV line and extracts
     * the individual fields, converting them to appropriate data types
     * and storing them in the provided ZipCodeRecord structure.
     *
     * Handles quoted fields and embedded commas correctly.
     */
    bool parseLine(const string& line, ZipCodeRecord& record);

    /**
     * @brief Splits a CSV line into individual fields
     * @param line The CSV line to split
     * @return Vector of strings containing individual fields
     *
     * This utility function correctly handles CSV formatting including:
     * - Quoted fields
     * - Embedded commas within quotes
     * - Leading/trailing whitespace
     */
    vector<string> splitCSV(const string& line);

    /**
     * @brief Trims whitespace from both ends of a string
     * @param str The string to trim
     * @return Trimmed string
     */
    string trim(const string& str);

public:
    /**
     * @brief Default constructor
     *
     * Creates an uninitialized ZipCodeBuffer. The open() method
     * must be called before reading any records.
     */
    ZipCodeBuffer();

    /**
     * @brief Parameterized constructor
     * @param csvFilename Path to the CSV file to open
     *
     * Creates a ZipCodeBuffer and automatically opens the specified file.
     * The header row is skipped during initialization.
     */
    explicit ZipCodeBuffer(const string& csvFilename);

    /**
     * @brief Destructor
     *
     * Ensures the file stream is properly closed when the object
     * is destroyed, preventing resource leaks.
     */
    ~ZipCodeBuffer();

    /**
     * @brief Opens a CSV file for reading
     * @param csvFilename Path to the CSV file
     * @return true if file opened successfully, false otherwise
     *
     * Opens the specified CSV file and skips the header row.
     * If a file is already open, it is closed first.
     */
    bool open(const string& csvFilename);

    /**
     * @brief Closes the currently open file
     *
     * Closes the file stream and resets internal state.
     * Safe to call even if no file is open.
     */
    void close();

    /**
     * @brief Checks if a file is currently open
     * @return true if file is open and ready for reading, false otherwise
     */
    bool isOpen() const;

    /**
     * @brief Reads the next ZIP code record from the file
     * @param record Reference to ZipCodeRecord to populate
     * @return true if a record was successfully read, false on EOF or error
     *
     * Reads one line from the CSV file, parses it, and populates the
     * provided ZipCodeRecord structure. Returns false when end of file
     * is reached or if a parsing error occurs.
     *
     * @note Automatically skips the header row on first read
     */
    bool readRecord(ZipCodeRecord& record);

    /**
     * @brief Reads and gathers all records from the file
     * @return Vector containing all ZIP code records from the file
     *
     * Reads the entire CSV file and returns all valid records as a vector.
     * The file position is reset to the beginning (after header) when complete.
     *
     * @note This method loads all data into memory - use with caution for very large files
     */
    vector<ZipCodeRecord> gatherAllRecords();

    /**
     * @brief Resets the file position to the beginning (after header)
     * @return true if reset was successful, false otherwise
     *
     * Seeks back to the start of the file and skips the header row again,
     * allowing the file to be re-read without closing and reopening.
     */
    bool reset();

    /**
     * @brief Gets the total number of records read so far
     * @return Count of records read
     */
    long getRecordCount() const;

    /**
     * @brief Gets the name of the currently open file
     * @return Filename string
     */
    string getFilename() const;
};

#include "ZipCodeBuffer.cpp"

#endif // ZIPCODEBUFFER_H
```

Index

∼ZipCodeBuffer
    ZipCodeBuffer, 6

calculateStateExtremes
    main.cpp, 20
close
    ZipCodeBuffer, 7
county
    ZipCodeRecord, 17

easternmost
    StateExtremes, 3

filename
    ZipCodeBuffer, 14
fileStream
    ZipCodeBuffer, 14

gatherAllRecords
    ZipCodeBuffer, 7
getFilename
    ZipCodeBuffer, 7
getRecordCount
    ZipCodeBuffer, 8

headerSkipped
    ZipCodeBuffer, 15

isOpen
    ZipCodeBuffer, 8

latitude
    ZipCodeRecord, 17
longitude
    ZipCodeRecord, 17

main
    main.cpp, 21
main.cpp, 19
    calculateStateExtremes, 20
    main, 21
    printStateExtremesTable, 21
maxLatitude
    StateExtremes, 3
maxLongitude
    StateExtremes, 3
minLatitude
    StateExtremes, 3
minLongitude
    StateExtremes, 3

northernmost
    StateExtremes, 4

open
    ZipCodeBuffer, 9

parseLine
    ZipCodeBuffer, 10
placeName
    ZipCodeRecord, 18
printStateExtremesTable
    main.cpp, 21

readRecord
    ZipCodeBuffer, 11
recordCount
    ZipCodeBuffer, 15
reset
    ZipCodeBuffer, 11

southernmost
    StateExtremes, 4
splitCSV
    ZipCodeBuffer, 12
state
    ZipCodeRecord, 18
StateExtremes, 1
    easternmost, 3
    maxLatitude, 3
    maxLongitude, 3
    minLatitude, 3
    minLongitude, 3
    northernmost, 4
    southernmost, 4
    StateExtremes, 2
    westernmost, 4

trim
    ZipCodeBuffer, 13

westernmost
    StateExtremes, 4

zipCode
    ZipCodeRecord, 18
ZipCodeBuffer, 5
    ∼ZipCodeBuffer, 6
    close, 7
    filename, 14
    fileStream, 14
    gatherAllRecords, 7
    getFilename, 7
    getRecordCount, 8
    headerSkipped, 15
    isOpen, 8

open, 9
parseLine, 10
readRecord, 11
recordCount, 15
reset, 11
splitCSV, 12
trim, 13
ZipCodeBuffer, 6
ZipCodeBuffer.cpp, 25
ZipCodeBuffer.h, 30
ZipCodeRecord, 15
county, 17
latitude, 17
longitude, 17
placeName, 18
state, 18
zipCode, 18
ZipCodeRecord, 16