Zip Code Group Project

Version 1.0

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

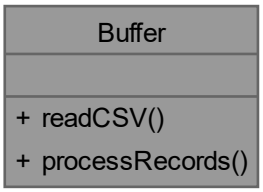
Class Documentation

3.1 Buffer Class Reference

A class to handle reading, processing, and validating zip code data.

`#include <Buffer.h>`

Collaboration diagram for Buffer:



Public Member Functions

- bool readCSV (const string &filename, vector< ZipCodeRecord > &records)

  Reads a CSV file, validates data integrity, and stores zip code records.
- void processRecords (const vector< ZipCodeRecord > &records, map< string, vector< ZipCodeRecord > > &state_map)

  Organizes zip code records by state.

3.1.1 Detailed Description

A class to handle reading, processing, and validating zip code data.

Definition at line 40 of file Buffer.h.

3.1.2 Member Function Documentation

3.1.2.1 readCSV()

```
bool Buffer::readCSV (
            const string & filename,
            vector< ZipCodeRecord > & records)
```

Reads a CSV file, validates data integrity, and stores zip code records.

Parameters

filename	The name of the CSV file to read.
records	A vector to store the read zip code records.

Returns

True if the file is read successfully, false otherwise.

This function reads a CSV file containing zip code data, extracts fields, checks for missing values, and stores valid records into a vector. It also prints warnings for missing non-critical fields.

Definition at line 10 of file Buffer.cpp.

3.1.2.2 processRecords()

```
void Buffer::processRecords (
            const vector< ZipCodeRecord > & records,
            map< string, vector< ZipCodeRecord > > & state_map)
```

Organizes zip code records by state.

Parameters

records	The vector of zip code records.
state_map	A map to store zip codes categorized by state.

This function groups zip code records by state into a map for easy retrieval.

Definition at line 68 of file Buffer.cpp.

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

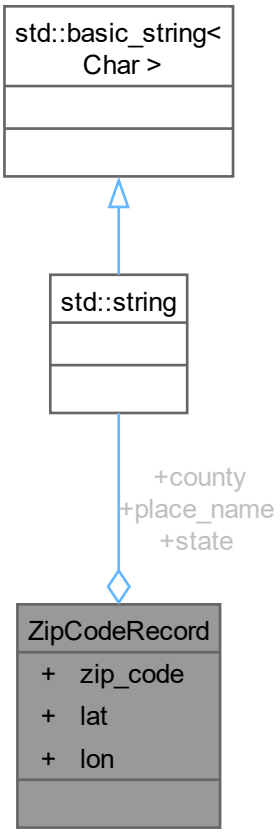
- Buffer.h
- Buffer.cpp

3.2 ZipCodeRecord Struct Reference

Structure to hold zip code data for a given location.

`#include <Buffer.h>`

Collaboration diagram for ZipCodeRecord:



Public Attributes

- int zip_code
  zip code of the location.
- string place_name
  City or place name.
- string state
  Two-letter state abbreviation.
- string county
  County name (can be empty).
- double lat
  Latitude coordinate of the location.
- double lon
  Longitude coordinate of the location.

3.2.1 Detailed Description

Structure to hold zip code data for a given location.

Definition at line 27 of file Buffer.h.

3.2.2 Member Data Documentation

3.2.2.1 zip_code

```
int ZipCodeRecord::zip_code
```

zip code of the location.

Definition at line 28 of file Buffer.h.

3.2.2.2 place_name

```
string ZipCodeRecord::place_name
```

City or place name.

Definition at line 29 of file Buffer.h.

3.2.2.3 state

```
string ZipCodeRecord::state
```

Two-letter state abbreviation.

Definition at line 30 of file Buffer.h.

3.2.2.4 county

```
string ZipCodeRecord::county
```

County name (can be empty).

Definition at line 31 of file Buffer.h.

3.2.2.5 lat

```
double ZipCodeRecord::lat
```

Latitude coordinate of the location.

Definition at line 32 of file Buffer.h.

3.2.2.6 lon

```
double ZipCodeRecord::lon
```

Longitude coordinate of the location.

Definition at line 33 of file Buffer.h.

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

- Buffer.h

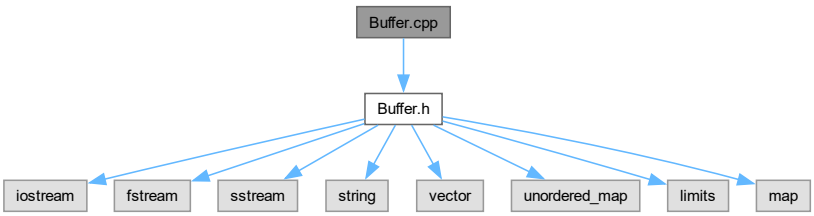
Chapter 4

File Documentation

4.1 Buffer.cpp File Reference

Implementation of the Buffer class for handling zip code data processing and validation.

`#include "Buffer.h"`
Include dependency graph for Buffer.cpp:



4.1.1 Detailed Description

Implementation of the Buffer class for handling zip code data processing and validation.

Definition in file Buffer.cpp.

4.2 Buffer.cpp

Go to the documentation of this file.

```
00001
00005
00006  #include "Buffer.h"
00007
00008  using namespace std;
00009
00010  bool Buffer::readCSV(const string& filename, vector<ZipCodeRecord>& records) {
00011      ifstream file(filename);
00012      if (!file.is_open()) {
```

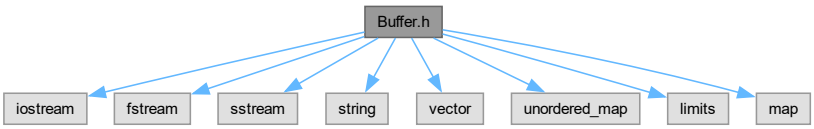
```
00013           cerr « "Error: Could not open the file " « filename « endl;
00014           return false;
00015       }
00016
00017       string line;
00018       getline(file, line); // Skip header
00019
00020       while (getline(file, line)) {
00021           stringstream ss(line);
00022           ZipCodeRecord record;
00023           string zip, lat, lon;
00024           vector<string> values;
00025           string token;
00026
00027           while (getline(ss, token, ',')) {
00028               values.push_back(token);
00029           }
00030
00031           if (values.size() != 6) {
00032               cerr « "Error: Incorrect number of columns on line: " « line « endl;
00033               continue;
00034           }
00035
00036           zip = values[0];
00037           record.place_name = values[1];
00038           record.state = values[2];
00039           record.county = values[3];
00040           lat = values[4];
00041           lon = values[5];
00042
00043           if (record.county.empty()) {
00044               cerr « "Warning: Missing county on line: " « line « endl;
00045           }
00046
00047           if (zip.empty() || record.state.empty() || lat.empty() || lon.empty()) {
00048               cerr « "Error: Missing critical values on line: " « line « endl;
00049               continue;
00050           }
00051
00052           try {
00053               record.zip_code = stoi(zip);
00054               record.lat = stod(lat);
00055               record.lon = stod(lon);
00056           } catch (const exception& e) {
00057               cerr « "Error parsing numeric values on line: " « line « " - " « e.what() « endl;
00058               continue;
00059           }
00060
00061           records.push_back(record);
00062       }
00063
00064       file.close();
00065       return true;
00066   }
00067
00068   void Buffer::processRecords(const vector<ZipCodeRecord>& records, map<string, vector<ZipCodeRecord»&
      state_map) {
00069       for (const auto& record : records) {
00070           state_map[record.state].push_back(record);
00071       }
00072   }
00073
```

4.3 Buffer.h File Reference

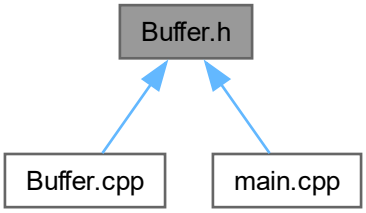
Header file for the Buffer class, which handles reading, processing, and validating zip code data.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <unordered_map>
#include <limits>
```

`#include <map>`
Include dependency graph for Buffer.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct ZipCodeRecord

  Structure to hold zip code data for a given location.
- class Buffer

  A class to handle reading, processing, and validating zip code data.

4.3.1 Detailed Description

Header file for the Buffer class, which handles reading, processing, and validating zip code data.

This class reads a CSV file containing zip codes, organizes the data by state, allows sorting by Zip Code or Place Name, and verifies data integrity before processing.

Definition in file Buffer.h.

4.4 Buffer.h

Go to the documentation of this file.

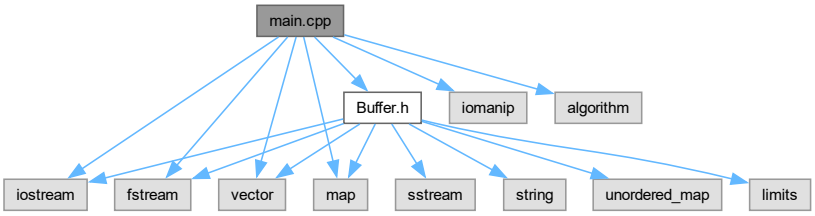
```
00001
00008
00009 #ifndef BUFFER_H
00010 #define BUFFER_H
00011
00012 #include <iostream>
00013 #include <fstream>
00014 #include <sstream>
00015 #include <string>
00016 #include <vector>
00017 #include <unordered_map>
00018 #include <limits>
00019 #include <map>
00020
00021 using namespace std;
00022
00027 struct ZipCodeRecord {
00028     int zip_code;
00029     string place_name;
00030     string state;
00031     string county;
00032     double lat;
00033     double lon;
00034 };
00035
00040 class Buffer {
00041 public:
00051     bool readCSV(const string& filename, vector<ZipCodeRecord>& records);
00052
00060     void processRecords(const vector<ZipCodeRecord>& records, map<string, vector<ZipCodeRecord>&
      state_map);
00061 };
00062
00063 #endif // BUFFER_H
```

4.5 main.cpp File Reference

Main program for processing zip code data and generating reports.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <map>
#include <iomanip>
#include <algorithm>
#include "Buffer.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

  Main function to process zip code data.

4.5.1 Detailed Description

Main program for processing zip code data and generating reports.

This program reads zip code data from a CSV file (us_postal_codes.csv), sorts it based on user selection (Zip Code or Place Name), and then organizes the data by state. It outputs the easternmost, westernmost, northernmost, and southernmost locations using either Zip Codes or Place Names.

Definition in file main.cpp.

4.5.2 Function Documentation

4.5.2.1 main()

```
int main ()
```

Main function to process zip code data.

Returns

0 on successful execution, -1 on error.

Definition at line 25 of file main.cpp.

4.6 main.cpp

Go to the documentation of this file.

```
00001
00010
00011 #include <iostream>
00012 #include <fstream>
00013 #include <vector>
00014 #include <map>
00015 #include <iomanip>
00016 #include <algorithm>
00017 #include "Buffer.h"
00018
00019 using namespace std;
00020
00025 int main() {
00026     vector<ZipCodeRecord> records;
00027     Buffer buffer;
00028     map<string, vector<ZipCodeRecord» state_map;
00029
00030     string filename = "us_postal_codes.csv";
00031
00032     // Ask user for sorting preference
00033     char sortChoice;
00034     while (true) {
00035         cout « "Do you want to sort by Zip Code (Z) or Place Name (P): ";
00036         cin » sortChoice;
00037         sortChoice = toupper(sortChoice);
00038
00039         if (sortChoice == 'Z' || sortChoice == 'P') {
00040             break;
00041         }
00042         cout « "Invalid choice! Please enter 'Z' for Zip Code or 'P' for Place Name.\n";
00043     }
00044
00045     // Read CSV file
00046     if (!buffer.readCSV(filename, records)) {
00047         cerr « "Error: Unable to read CSV file: " « filename « endl;
00048         return -1;
00049     }
```

```
00050
00051     // Sort data based on user choice
00052     if (sortChoice == 'Z') {
00053         sort(records.begin(), records.end(), [](const ZipCodeRecord& a, const ZipCodeRecord& b) {
00054             return a.zip_code < b.zip_code;
00055         });
00056     } else {
00057         sort(records.begin(), records.end(), [](const ZipCodeRecord& a, const ZipCodeRecord& b) {
00058             return a.place_name < b.place_name;
00059         });
00060     }
00061
00062     buffer.processRecords(records, state_map);
00063
00064     ofstream outfile_txt("SortedLocations.txt");
00065     ofstream outfile_csv("SortedLocations.csv");
00066
00067     // Adjust column widths
00068     int stateWidth = 5;
00069     int fieldWidth = (sortChoice == 'Z') ? 12 : 20;  // Zip codes = 12 width, Place names = 20 width
00070
00071     // Print headers
00072     outfile_txt « left « setw(stateWidth) « "State" « " | "
00073                 « setw(fieldWidth) « "Easternmost" « " | "
00074                 « setw(fieldWidth) « "Westernmost" « " | "
00075                 « setw(fieldWidth) « "Northernmost" « " | "
00076                 « setw(fieldWidth) « "Southernmost" « " |\n";
00077
00078     outfile_txt « string((6 + (fieldWidth + 3) * 4) + 1, '=') « "\n";
00079
00080     outfile_csv « "State,Easternmost,Westernmost,Northernmost,Southernmost\n";
00081
00082     // Process each state and determine extreme locations
00083     for (const auto& entry : state_map) {
00084         const string& state = entry.first;
00085         const vector<ZipCodeRecord>& zipRecords = entry.second;
00086
00087         string eastPlace, westPlace, northPlace, southPlace;
00088         int eastZip, westZip, northZip, southZip;
00089         double minLon = numeric_limits<double>::max();
00090         double maxLon = numeric_limits<double>::lowest();
00091         double maxLat = numeric_limits<double>::lowest();
00092         double minLat = numeric_limits<double>::max();
00093
00094         for (const auto& record : zipRecords) {
00095             if (record.lon < minLon) {
00096                 minLon = record.lon;
00097                 eastZip = record.zip_code;
00098                 eastPlace = record.place_name;
00099             }
00100             if (record.lon > maxLon) {
00101                 maxLon = record.lon;
00102                 westZip = record.zip_code;
00103                 westPlace = record.place_name;
00104             }
00105             if (record.lat > maxLat) {
00106                 maxLat = record.lat;
00107                 northZip = record.zip_code;
00108                 northPlace = record.place_name;
00109             }
00110             if (record.lat < minLat) {
00111                 minLat = record.lat;
00112                 southZip = record.zip_code;
00113                 southPlace = record.place_name;
00114             }
00115         }
00116
00117         if (sortChoice == 'Z') {
00118             // Output using zip Codes
00119             outfile_txt « left « setw(stateWidth) « state « " | "
00120                         « right « setw(fieldWidth) « eastZip « " | "
00121                         « setw(fieldWidth) « westZip « " | "
00122                         « setw(fieldWidth) « northZip « " | "
00123                         « setw(fieldWidth) « southZip « " |\n";
00124
00125             outfile_csv « state « "," « eastZip « "," « westZip « "," « northZip « "," « southZip «
      "\n";
00126         } else {
00127             // Output using Place Names
00128             outfile_txt « left « setw(stateWidth) « state « " | "
00129                         « left « setw(fieldWidth) « eastPlace « " | "
00130                         « left « setw(fieldWidth) « westPlace « " | "
00131                         « left « setw(fieldWidth) « northPlace « " | "
00132                         « left « setw(fieldWidth) « southPlace « " |\n";
00133
00134             outfile_csv « state « "," « eastPlace « "," « westPlace « "," « northPlace « "," «
      southPlace « "\n";
```

```
00135         }
00136     }
00137
00138     // Close files
00139     outfile_txt.close();
00140     outfile_csv.close();
00141     cout « "Output written to SortedLocations.txt and SortedLocations.csv\n";
00142     return 0;
00143 }
```

Index