Zip Code Group Project 2.0

Generated by Doxygen 1.9.2

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

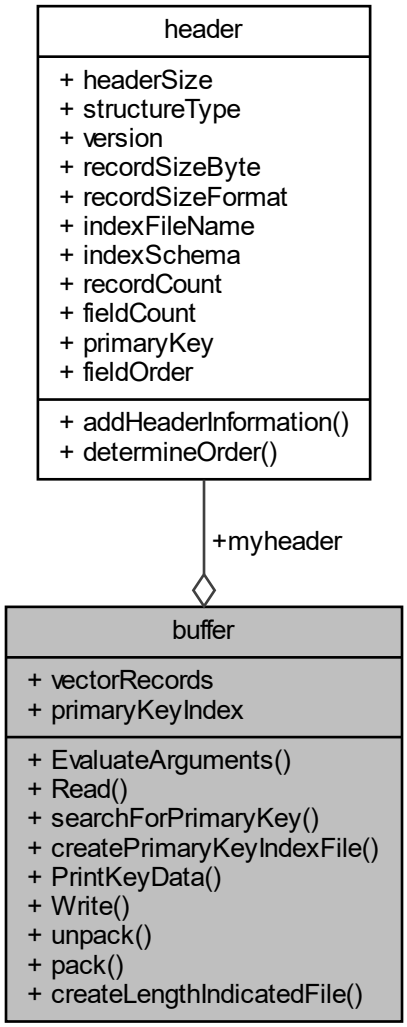
Class Documentation

3.1 buffer Class Reference

class that stores data into a vector and reads, writes, packs and unpacks data

`#include <buffer.h>`

Collaboration diagram for buffer:



Public Member Functions

- void EvaluateArguments (string arr[ ], int count)

  This function takes in the command line arguments given by the user and then turns them into usable zipcode integers. then it searches the primary key for said zipcode and checks for existence then creates primary key index file.
- void Read (string file)

  Reads in file name and then reads the file and its information. distributes the various information to the different functions.
- int searchForPrimaryKey (primaryKey)

  Searches to see if the primary key is in the file, if so, returns the location.
- void createPrimaryKeyIndexFile ()

  Creates the primary key index file and stores the keys as well as their locations.
- void PrintKeyData ()

opens file to search for key data and then displays it if found

- void Write ()

  Sorts and Outputs the most North,South,West and East Zip code for each state.

- record unpack (string)

  Unpacks record information line by line.

- void pack (record)

  Packs each record and puts them into the vector.

- void createLengthIndicatedFile (string)

  Creates the length indicated file with the header at the top.

Public Attributes

- vector< record > vectorRecords
- vector< primaryKey > primaryKeyIndex
- header myheader

Friends

- bool operator< (const record &r1, const record &r2)

3.1.1 Detailed Description

class that stores data into a vector and reads, writes, packs and unpacks data

Author

Evan Burdick, Joseph Kuzko, Matthew Xiong, Jordan Knight

Definition at line 41 of file buffer.h.

3.1.2 Member Function Documentation

3.1.2.1 createLengthIndicatedFile()

```
void buffer::createLengthIndicatedFile (
            string headerRecord )
```

Creates the length indicated file with the header at the top.

Author

Jordan Knight

Parameters

A	string

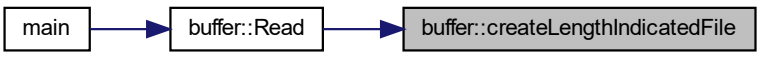
Precondition

Records have been read in

Definition at line 57 of file buffer.cpp.

```
00057                                                    {
00058
00059     record temp_record;
00060     ofstream fileout;
00061     fileout.open("lengthIndicated.txt"); //open the file
00062     fileout « headerRecord;
00063     for (int i = 0; i < myheader.recordCount; i++) //for each record, print the record to the file
00064     {
00065         temp_record = vectorRecords[i];
00066         fileout « temp_record.recordSize« ","; //first, output the length of the record
00067         for (int i = 0; i < myheader.fieldCount; i++)
00068         {
00069             if(myheader.fieldOrder[i] == 1){
00070                 fileout «temp_record.zipcode « ",";
00071             }
00072             else if (myheader.fieldOrder[i] == 2){
00073                 fileout «temp_record.city « ",";
00074             }
00075             else if (myheader.fieldOrder[i] == 3){
00076                 fileout «temp_record.state « ",";
00077             }
00078             else if (myheader.fieldOrder[i] == 4){
00079                 fileout «temp_record.county « ",";
00080             }
00081             else if (myheader.fieldOrder[i] == 5){
00082                 fileout «temp_record.latitude « ",";
00083             }
00084             else if (myheader.fieldOrder[i] == 6){
00085                 fileout «temp_record.longitude « ",";
00086             }
00087         }
00088     }
00089     fileout.close(); //close the file before leaving
00090 }
```

Here is the caller graph for this function:



3.1.2.2 createPrimaryKeyIndexFile()

```
void buffer::createPrimaryKeyIndexFile ( )
```

Creates the primary key index file and stores the keys as well as their locations.

Author

Evan Burdick

Precondition

Records have been read in

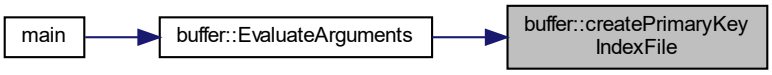
Postcondition

the primary key index file has been made or an error message is displayed

Definition at line 157 of file buffer.cpp.

```
00157                                       {
00158     ofstream outfile;
00159     outfile.open(myheader.indexFileName); //open the file
00160     primaryKey k;
00161
00162     outfile « "Key Locations marked as \"-1\" do not exist in the csv data file."«endl;
00163     outfile « string(70, '-') «endl;
00164     for (int i = 0; i < primaryKeyIndex.size(); i++)
00165     {
00166         k = primaryKeyIndex[i];
00167         outfile« left « setw(10)« "PrimaryKey: " « left « setw(6)«k.key « left« setw(1)« " | "« left «
      setw(10)«"Location: "« k.byteLocation « endl; //print key index to file
00168     }
00169     outfile.close(); //close the file
00170 }
```

Here is the caller graph for this function:



3.1.2.3 EvaluateArguments()

```
void buffer::EvaluateArguments (
             string arr[],
             int count )
```

This function takes in the command line arguments given by the user and then turns them into usable zipcode integers. then it searches the primary key for said zipcode and checks for existence then creates primary key index file.

Author

Evan Burdick

Parameters

1	command line args
2	Count is the number of arguments

Precondition

there is arguments taken in from command line and the count of how many command line arguments there are

Definition at line 135 of file buffer.cpp.

```
00135                                                   {
00136     primaryKey k;
00137     int zipcode, location;
00138     string argument;
00139     for (int i = 0; i < count; i++) //for every command argument after the first (which is the
      filename)
00140     {
00141         argument = arg[i];
00142         if (argument[0] == '-') //argument prefix
00143         {
00144             if (argument[1] =='Z'|| argument[1]== 'z') //zip code argument
00145             {
00146                 argument.erase(0,2); //erase the -z from the argument
00147                 zipcode = stoi(argument); //typecast the string zipcode into an integer
00148                 k.key = zipcode;
00149                 location = searchForPrimaryKey(k); //set the location of key if it exists,
00150                 k.byteLocation = location;
00151                 primaryKeyIndex.push_back(k); //push the key into the vector of primary keys
00152             }
00153         }
00154     }
00155     createPrimaryKeyIndexFile();
00156 }
```

Here is the caller graph for this function:

3.1.2.4 pack()

```
void buffer::pack (
            record new_record )
```

Packs each record and puts them into the vector.

Author

Jordan Knight

Parameters

Object	of myrecord (structured line of data) passed in

Precondition

Must be an object of class record created and passed in

Postcondition

New record has been added into the vector

Definition at line 131 of file buffer.cpp.

```
00131                                   {
00132
00133     vectorRecords.push_back(new_record); //add record to the vector
00134 }
```

Here is the caller graph for this function:

3.1.2.5 PrintKeyData()

```
void buffer::PrintKeyData ( )
```

opens file to search for key data and then displays it if found

Author

Matthew Xiong

Precondition

there is zipcode data to print that was found

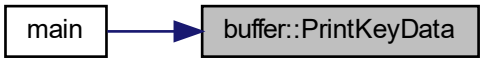
Postcondition

displays a data table for the user to see the results of the search

Definition at line 253 of file buffer.cpp.

```
00253                           {
00254     ifstream inFile;
00255     inFile.open("lengthIndicated.txt");
00256     string line;
00257     const int design = 140; //value for formatting output
00258     primaryKey k;
00259     bool falseKey = false;
00260     getline(inFile, line); //ignore the first line, which contains header information
00261 
00262     cout « string(design, '-') « endl; //header for output
00263     cout « left« setw(12)« "Zipcode"« left«setw(40)« "City" « left«setw(12)« "State" « left«setw(40)«
      "County" « left«setw(15)« "Latitude" « left«setw(15)« "Longitude" « endl; //header for output
00264     cout « string(design, '-') « endl; //header for output
00265 
00266     for (int i = 0; i < primaryKeyIndex.size(); i++) //for every primarykey in the index
00267     {
00268         k = primaryKeyIndex[i];
00269         if(k.byteLocation == -1){
00270             falseKey = true;
00271         }
00272         for (int i = 0; i < myheader.recordCount; i++)
00273         {
00274             record r = vectorRecords[i];
00275             if(stoi(r.zipcode) == k.key) //found the key and recorded location
00276             {
00277                 cout « left« setw(12)« r.zipcode« left«setw(40)« r.city« left«setw(12)« r.state«
      left«setw(40)« r.county « left«setw(15)« r.latitude« left«setw(15)« r.longitude « endl;
00278             }
00279         }
00280         inFile.close();
00281     }
00282     if (falseKey) //there was an incorrect key found, output all of the incorrect keys
00283     {
00284         cout « "\n\n\nThese keys did not exist in the file: ";
00285         for (int i = 0; i < primaryKeyIndex.size(); i++)
00286         {
00287             k = primaryKeyIndex[i];
00288             if(k.byteLocation == -1) //the location was not found for this key
00289             {
00290                 cout «" " «k.key « ","; //output the key
00291             }
00292         }
00293     }
00294 }
```

Here is the caller graph for this function:



3.1.2.6 Read()

```
void buffer::Read (
            string file )
```

Reads in file name and then reads the file and its information. distributes the various information to the different functions.

Author

Joseph Kuzko

Parameters

String	of the file name

Precondition

there is a file name to open

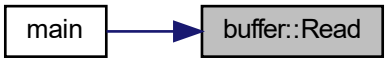
Postcondition

Data has been read into various functions

Definition at line 9 of file buffer.cpp.

```
00010 {
00011
00012     record myrecord;
00013     string headerRecordString ="", line, tempString;
00014     string columnHeaders; //concatenation of all of the column headers, to determine their order
00015     char c;
00016     int commaCount = 0; //keep track of number of commas in data (to determine how many fields/records
      there are)
00017
00018     ifstream filein;
00019     filein.open(file);
00020
00021     if(!filein)
00022     {
00023         cout « "The file you have entered does not exist, re-run the program with the correct file
      name"«endl;
00024         abort();
00025     }
00026     for (int i = 0; i < 3; i++) //gets the first 3 lines of file, which contain information on how the
      columns are ordered
00027     {
00028         getline(filein, line);
00029
00030         if(line == ""){
00031             cout « "The file you have entered is empty, please re-run the program with a correct
      file"« endl;
00032             abort();
00033         }
00034          columnHeaders = columnHeaders + line;
00035     }
00036
00037     myheader.determineOrder(columnHeaders); //determines the order of the columns
00038
00039     while (!filein.eof())
00040     {
00041         getline(filein, line);
00042         if (line != "")
00043         {
00044             myrecord = unpack(line);
00045             pack(myrecord);
00046         }
00047     }
00048
00049     filein.close(); //close the file
00050
00051     myheader.recordCount = vectorRecords.size();
00052
00053     headerRecordString = myheader.addHeaderInformation(); // get header record string
00054
00055     createLengthIndicatedFile(headerRecordString);
00056 }
```

Here is the caller graph for this function:



3.1.2.7 searchForPrimaryKey()

```
int buffer::searchForPrimaryKey (
            primaryKey k )
```

Searches to see if the primary key is in the file, if so, returns the location.

Author

Evan Burdick

Parameters

The	primary key 'k' thats being searched for

Precondition

there is a primary key to look for

Postcondition

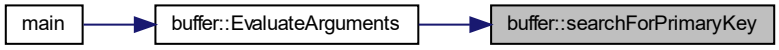
returns either -1 which means not found or returns the location where its found

Definition at line 171 of file buffer.cpp.

```
00171                                           { //searches to see if the primary key is in the file, if
       so, returns the location
00172
00173     string line;
00174     ifstream filein;
00175     filein.open("lengthIndicated.txt"); //open the file
00176     int location =0,totalLocation = 0;
00177     string numberString;
00178     getline(filein, line); //skip first line which contains the header information
00179
00180         for (int i = 0; i < myheader.recordCount; i++)
00181         {
00182             record r = vectorRecords[i];
00183             location = r.recordSize;
00184             if(stoi(r.zipcode) == k.key) //found the key and recorded location
00185             {
00186                 filein.close(); //close the file
00187                 return(totalLocation); //return the found key location
00188             }
00189             totalLocation = totalLocation + location; //update the total location
```

```
00190         }
00191             filein.close(); //close the file
00192             return(-1); //key was not found, set to negative number to indicate it wasn't
00193 }
```

Here is the caller graph for this function:



3.1.2.8 unpack()

```
record buffer::unpack (
             string line )
```

Unpacks record information line by line.

Author

Joseph Kuzko

Parameters

A	string which is a line of data

Precondition

There is information that has been packed into record

Postcondition

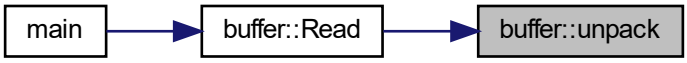
A line of data has been unpacked into temp_record

Definition at line 92 of file buffer.cpp.

```
00092                                {
00093     record temp_record;
00094     temp_record.recordSize = line.length() + myheader.recordSizeByte +2;
00095     int temp_count = 0, switchTemp;
00096     string record[myheader.fieldCount];
00097     for (int i = 0; i <line.length(); i++) { //for every character in line
00098
00099         if (line[i] == ','|| line[i-1] ==',') {
00100             temp_count++;
00101             i++;
00102         }
00103         switchTemp = myheader.fieldOrder[temp_count];
00104
00105         switch (switchTemp)
00106         {
00107         case 1:
00108             temp_record.zipcode += line[i];
```

```
00109             break;
00110         case 2:
00111             temp_record.city += line[i];
00112             break;
00113         case 3:
00114             temp_record.state += line[i];
00115             break;
00116         case 4:
00117             temp_record.county += line[i];
00118             break;
00119         case 5:
00120             temp_record.latitude += line[i];
00121             break;
00122         case 6:
00123             temp_record.longitude += line[i];
00124             break;
00125         }
00126     }
00127     return temp_record;
00128 }
```

Here is the caller graph for this function:



3.1.2.9 Write()

```
void buffer::Write ( )
```

Sorts and Outputs the most North,South,West and East Zip code for each state.

Author

: Matthew Xiong

Precondition

Data has been read in and packed into the vector

Postcondition

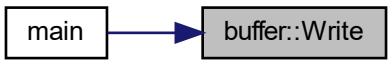
Data has been sorted and put into a table

Definition at line 195 of file buffer.cpp.

```
00195                   {
00196
00197     sort(vectorRecords.begin(), vectorRecords.end()); //sort the vector to make processing easier
00198
00199 // Evaluate and ouput west,east,north, and south most zip codes for each state //
00200     record myrecord;
00201     myrecord = vectorRecords[0];
00202     string st = myrecord.state;
```

```
00203     const double reset = -999999; //value to restore westmost, eastmost, northmost, southmost values
      after every entry per state has been accounted for
00204     const int design = 55; //value for formatting output
00205
00206     double westMost = reset, eastMost = reset;
00207     double northMost = reset, southMost = reset; //stores the (west,east,north,south) most value for
      each state. Set to impossible value to start.
00208     string zipWestMost, zipEastMost, zipNorthMost, zipSouthMost; //stores the zip code for the
      (west,east,north,south) most values.
00209
00210     cout « string(design, '-') « endl; //header for output
00211     cout « setw(7) « "State"« setw(12)« "North-Most" « setw(12)« "South-Most" « setw(12)« "East-Most"
      « setw(12)« "West-Most" « endl; //header for output
00212     cout « string(design, '-')« endl; //header for output
00213
00214     for(int i = 0; i < vectorRecords.size(); i++) //for every record in the vector
00215     {
00216         myrecord = vectorRecords[i]; //set myrecord equal to current record being read
00217         if(st == myrecord.state){ //the same state is being read
00218             if(stod(myrecord.longitude) < westMost || westMost == reset) //check for west most
00219             {
00220                 westMost = stod(myrecord.longitude); //typecast the longitude and store it as the
      westmost value
00221                 zipWestMost = myrecord.zipcode; //store the zipcode for this record
00222             }
00223             if(stod(myrecord.longitude) > eastMost || eastMost == reset) //check for east most
00224             {
00225                 eastMost = stod(myrecord.longitude); //typecast the longitude and store it as the
      eastmost value
00226                 zipEastMost = myrecord.zipcode; //store the zipcode for this record
00227             }
00228             if(stod(myrecord.latitude) > northMost || northMost == reset) //check for south most
00229             {
00230                 northMost = stod(myrecord.latitude); //typecast the latitude and store it as the
      northmost value
00231                 zipNorthMost = myrecord.zipcode; //store the zipcode for this record
00232             }
00233             if(stod(myrecord.latitude) < southMost || southMost == reset) //check for north most
00234             {
00235                 southMost = stod(myrecord.latitude); //typecast the latitude and store it as the
      southmost value
00236                 zipSouthMost = myrecord.zipcode; //store the zipcode for this record
00237             }
00238         }
00239         else{
00240             cout « setw(7) « vectorRecords[i-1].state« setw(12)« zipNorthMost « setw(12)« zipSouthMost
      « setw(12)« zipEastMost « setw(12)« zipWestMost « endl; //output the info for the state
00241
00242             st = myrecord.state;
00243             westMost = reset; //reset to impossible value
00244             eastMost = reset; //reset to impossible value
00245             northMost = reset; //reset to impossible value
00246             southMost = reset; //reset to impossible value
00247             i--; //decrement so that this current state is accounted for
00248         }
00249     }
00250      cout « string(design, '-')« endl; //footer for output
00251
00252 }
```

Here is the caller graph for this function:



3.1.3 Friends And Related Function Documentation

3.1.3.1 operator<

```
bool operator< (
            const record & r1,
            const record & r2 )  [friend]
```

Definition at line 295 of file buffer.cpp.

```
00295                                                 {
00296     return r1.state < r2.state;  //compare the state of two records, to help with sorting
00297 }
```

3.1.4 Member Data Documentation

3.1.4.1 myheader

```
header buffer::myheader
```

Definition at line 47 of file buffer.h.

3.1.4.2 primaryKeyIndex

```
vector<primaryKey> buffer::primaryKeyIndex
```

Definition at line 46 of file buffer.h.

3.1.4.3 vectorRecords

```
vector<record> buffer::vectorRecords
```

Definition at line 45 of file buffer.h.

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

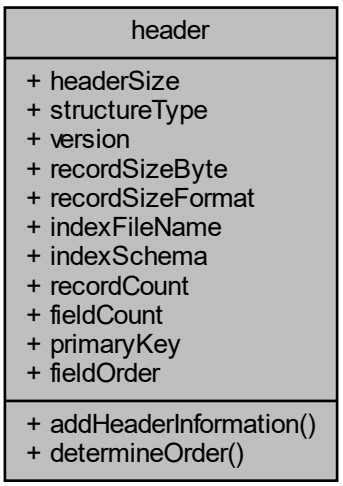
- buffer.h
- buffer.cpp

3.2 header Class Reference

class for the structure of the header

`#include <header.h>`

Collaboration diagram for header:



Public Member Functions

- string addHeaderInformation ()

  this function creates the header string and returns it
- void determineOrder (string)

  takes in a string of column names then determines the order to read the information and creates a vector for it

Public Attributes

- int headerSize
- string structureType
- double version
- int recordSizeByte = 2
- string recordSizeFormat
- string indexFileName
- string indexSchema
- int recordCount
- int fieldCount = 6
- int primaryKey
- int fieldOrder [ ]

3.2.1 Detailed Description

class for the structure of the header

Author

Evan Burdick

Definition at line 18 of file header.h.

3.2.2 Member Function Documentation

3.2.2.1 addHeaderInformation()

```
string header::addHeaderInformation ( )
```

this function creates the header string and returns it

Author

Evan Burdick

Precondition

there is a file to be read with a header

Definition at line 8 of file header.cpp.

```
00009 {
00010     string hInfo; //stores all header information, returns at the end of function.
00011
00012     structureType = "Length Indicated Records, Comma Seperated fields";
00013     version = 2.0;
00014     // headerSize determined after seeing length of header
00015     // recordSizeByte hard coded in header.h
00016     recordSizeFormat = "ASCII";
00017     indexFileName = "indexFile.txt";
00018     indexSchema = "Zipcode, ByteLocation";
00019     //fieldCount hard coded in header.h
00020     //record cound set in buffer::Read()
00021     //primaryKey set in determineOrder
00022     //fieldOrder set in determineOrder
00023
00024     hInfo = "Structure: " + structureType + ","+
00025             "Version: " + to_string(version) + ","+
00026             "Record Size Byte: " + to_string(recordSizeByte) + ","+
00027             "Record Size Format: " + recordSizeFormat + ","+
00028             "Index File Name: " + indexFileName + ","+
00029             "Index Schema: " + indexSchema + ","+
00030             "Number of fields in each record: " + to_string(fieldCount) + ","+
00031             "Number of records in file: " + to_string(recordCount) + ","+
00032             "Primary Key: " + to_string(primaryKey) + ","+
00033             "Field Order: [" + to_string(fieldOrder[0]) + ","+ to_string(fieldOrder[1]) + ","+
      to_string(fieldOrder[2]) + ","+ to_string(fieldOrder[3]) + ","+ to_string(fieldOrder[4]) + ","+
      to_string(fieldOrder[5]) + "]\n";
00034
00035     headerSize = hInfo.size();
00036     string headerSizeString = to_string(headerSize);
00037     headerSize = headerSize + headerSizeString.length(); //set headerSize equal to the contents of the
      header, the number of bytes the headerSize takes, and + 1 for the comma included after the size.
00038
00039     hInfo.insert(0, to_string(headerSize) + ","); //insert the headersize to the beginning of the
      header
```

```
00040     return hInfo; //return the header string
00041 }
```

Here is the caller graph for this function:

3.2.2.2 determineOrder()

```
void header::determineOrder (
             string columnHeaders )
```

takes in a string of column names then determines the order to read the information and creates a vector for it

Author

Evan Burdick

Parameters

string	of te column names

Precondition

there is column names to be read

Postcondition

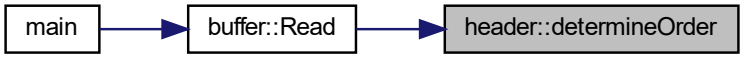
creates a vector in which the columns are to be read for the splitting of information

Definition at line 43 of file header.cpp.

```
00044 {
00045     string label ="";
00046     int j = 0;
00047     for (int i = 0; i < columnHeaders.size(); i++) //for every character in the column header
00048     {
00049         if(columnHeaders[i] != ',') //read in characters until it forms a readable label
00050         {
00051             label = label + columnHeaders[i];
00052         }
00053         if(label == "\"ZipCode\""){
00054             fieldOrder[j] = 1;
00055             primaryKey = j+1;
00056             j++;
00057             label = ""; //reset label
00058         }
00059         else if(label == "\"PlaceName\""){
00060             fieldOrder[j] = 2;
00061             j++;
```

```
00062             label = ""; //reset label
00063         }
00064         else if(label == "State"){
00065             fieldOrder[j] = 3;
00066             j++;
00067             label = ""; //reset label
00068         }
00069         else if(label == "County"){
00070             fieldOrder[j] = 4;
00071             j++;
00072             label = ""; //reset label
00073         }
00074         else if(label == "Lat"){
00075             fieldOrder[j] = 5;
00076             j++;
00077             label = ""; //reset label
00078         }
00079         else if(label == "Long"){
00080             fieldOrder[j] = 6;
00081             j++;
00082             label = ""; //reset label
00083         }
00084     }
00085 }
```

Here is the caller graph for this function:



3.2.3 Member Data Documentation

3.2.3.1 fieldCount

```
int header::fieldCount = 6
```

Definition at line 47 of file header.h.

3.2.3.2 fieldOrder

```
int header::fieldOrder[]
```

Definition at line 49 of file header.h.

3.2.3.3 headerSize

```
int header::headerSize
```

Definition at line 39 of file header.h.

3.2.3.4 indexFileName

```
string header::indexFileName
```

Definition at line 44 of file header.h.

3.2.3.5 indexSchema

```
string header::indexSchema
```

Definition at line 45 of file header.h.

3.2.3.6 primaryKey

```
int header::primaryKey
```

Definition at line 48 of file header.h.

3.2.3.7 recordCount

```
int header::recordCount
```

Definition at line 46 of file header.h.

3.2.3.8 recordSizeByte

```
int header::recordSizeByte = 2
```

Definition at line 42 of file header.h.

3.2.3.9 recordSizeFormat

```
string header::recordSizeFormat
```

Definition at line 43 of file header.h.

3.2.3.10 structureType

```
string header::structureType
```

Definition at line 40 of file header.h.

3.2.3.11 version

```
double header::version
```

Definition at line 41 of file header.h.

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

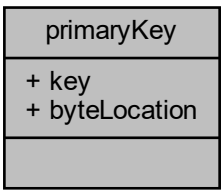
- header.h
- header.cpp

3.3 primaryKey Class Reference

class for the structure of the header

```
#include <primarykey.h>
```

Collaboration diagram for primaryKey:



Public Attributes

- int key
- int byteLocation

3.3.1 Detailed Description

class for the structure of the header

Author

Evan Burdick

Definition at line 16 of file primarykey.h.

3.3.2 Member Data Documentation

3.3.2.1 byteLocation

```
int primaryKey::byteLocation
```

Definition at line 19 of file primarykey.h.

3.3.2.2 key

```
int primaryKey::key
```

Definition at line 18 of file primarykey.h.

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

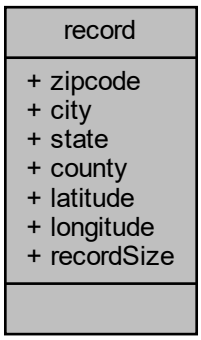
- primarykey.h

3.4 record Class Reference

class for the structure of data stored as strings, and the size of the record stored as an integer

`#include <buffer.h>`

Collaboration diagram for record:



Public Attributes

- string zipcode
- string city
- string state
- string county
- string latitude
- string longitude
- int recordSize

3.4.1 Detailed Description

class for the structure of data stored as strings, and the size of the record stored as an integer

Author

Joseph Kuzko,

Definition at line 23 of file buffer.h.

3.4.2 Member Data Documentation

3.4.2.1 city

```
string record::city
```

Definition at line 27 of file buffer.h.

3.4.2.2 county

```
string record::county
```

Definition at line 29 of file buffer.h.

3.4.2.3 latitude

```
string record::latitude
```

Definition at line 30 of file buffer.h.

3.4.2.4 longitude

```
string record::longitude
```

Definition at line 31 of file buffer.h.

3.4.2.5 recordSize

```
int record::recordSize
```

Definition at line 32 of file buffer.h.

3.4.2.6 state

```
string record::state
```

Definition at line 28 of file buffer.h.

3.4.2.7 zipcode

```
string record::zipcode
```

Definition at line 26 of file buffer.h.

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

- buffer.h

Chapter 4

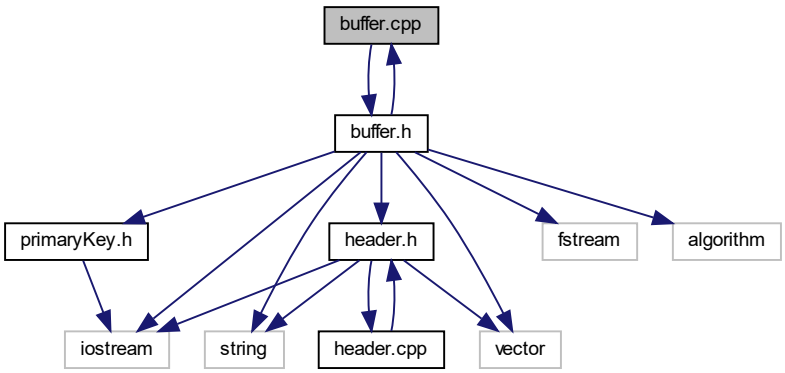
File Documentation

4.1 buffer.cpp File Reference

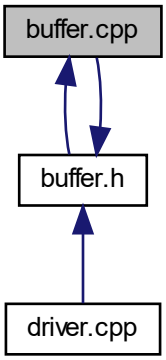
The file that drives the program essentially, reads in file and makes proper calls to pack/unpack.

```
#include "buffer.h"
```

Include dependency graph for buffer.cpp:



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



Functions

- bool operator< (const record &r1, const record &r2)

4.1.1 Detailed Description

The file that drives the program essentially, reads in file and makes proper calls to pack/unpack.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file buffer.cpp.

4.1.2 Function Documentation

4.1.2.1 operator<()

```
bool operator< (
            const record & r1,
            const record & r2 )
```

Definition at line 295 of file buffer.cpp.

```
00295                                                {
00296     return r1.state < r2.state;  //compare the state of two records, to help with sorting
00297 }
```

4.2 buffer.cpp

Go to the documentation of this file.

```
00001 //---------------------------------------------------------------------------
00005 //---------------------------------------------------------------------------
00006 #include "buffer.h"
00007
00008 //see header for info
00009 void buffer::Read(string file)
00010 {
00011
00012     record myrecord;
00013     string headerRecordString ="", line, tempString;
00014     string columnHeaders; //concatenation of all of the column headers, to determine their order
00015     char c;
00016     int commaCount = 0; //keep track of number of commas in data (to determine how many fields/records
      there are)
00017
00018     ifstream filein;
00019     filein.open(file);
00020
00021     if(!filein)
00022     {
00023         cout « "The file you have entered does not exist, re-run the program with the correct file
      name"«endl;
00024         abort();
00025     }
00026     for (int i = 0; i < 3; i++) //gets the first 3 lines of file, which contain information on how the
      columns are ordered
00027     {
00028         getline(filein, line);
00029
00030         if(line == ""){
00031             cout « "The file you have entered is empty, please re-run the program with a correct
      file"« endl;
00032             abort();
00033         }
00034          columnHeaders = columnHeaders + line;
00035     }
00036
00037     myheader.determineOrder(columnHeaders); //determines the order of the columns
00038
00039     while (!filein.eof())
00040     {
00041         getline(filein, line);
00042         if (line != "")
00043         {
00044             myrecord = unpack(line);
00045             pack(myrecord);
00046         }
00047     }
00048
00049     filein.close(); //close the file
00050
00051     myheader.recordCount = vectorRecords.size();
00052
00053     headerRecordString = myheader.addHeaderInformation(); // get header record string
00054
00055     createLengthIndicatedFile(headerRecordString);
00056 }
00057 void buffer::createLengthIndicatedFile(string headerRecord) {
00058
00059     record temp_record;
00060     ofstream fileout;
00061     fileout.open("lengthIndicated.txt"); //open the file
00062     fileout « headerRecord;
00063     for (int i = 0; i < myheader.recordCount; i++) //for each record, print the record to the file
00064     {
00065         temp_record = vectorRecords[i];
00066         fileout « temp_record.recordSize« ","; //first, output the length of the record
00067         for (int i = 0; i < myheader.fieldCount; i++)
00068         {
00069             if(myheader.fieldOrder[i] == 1){
00070                 fileout «temp_record.zipcode « ",";
00071             }
00072             else if (myheader.fieldOrder[i] == 2){
00073                 fileout «temp_record.city « ",";
00074             }
00075             else if (myheader.fieldOrder[i] == 3){
00076                 fileout «temp_record.state « ",";
00077             }
00078             else if (myheader.fieldOrder[i] == 4){
00079                 fileout «temp_record.county « ",";
00080             }
00081             else if (myheader.fieldOrder[i] == 5){
```

```
00082                 fileout «temp_record.latitude « ",";
00083             }
00084             else if (myheader.fieldOrder[i] == 6){
00085                 fileout «temp_record.longitude « ",";
00086             }
00087         }
00088     }
00089     fileout.close(); //close the file before leaving
00090 }
00091 //see header for info
00092 record buffer::unpack(string line) {
00093     record temp_record;
00094     temp_record.recordSize = line.length() + myheader.recordSizeByte +2;
00095     int temp_count = 0, switchTemp;
00096     string record[myheader.fieldCount];
00097     for (int i = 0; i <line.length(); i++) { //for every character in line
00098
00099         if (line[i] == ',' || line[i-1] ==',') {
00100             temp_count++;
00101             i++;
00102         }
00103         switchTemp = myheader.fieldOrder[temp_count];
00104
00105         switch (switchTemp)
00106         {
00107         case 1:
00108             temp_record.zipcode += line[i];
00109             break;
00110         case 2:
00111             temp_record.city += line[i];
00112             break;
00113         case 3:
00114             temp_record.state += line[i];
00115             break;
00116         case 4:
00117             temp_record.county += line[i];
00118             break;
00119         case 5:
00120             temp_record.latitude += line[i];
00121             break;
00122         case 6:
00123             temp_record.longitude += line[i];
00124             break;
00125         }
00126     }
00127     return temp_record;
00128 }
00129
00130 //see header for info
00131 void buffer::pack(record new_record) {
00132
00133     vectorRecords.push_back(new_record); //add record to the vector
00134 }
00135 void buffer::EvaluateArguments(string arg[], int count) {
00136     primaryKey k;
00137     int zipcode, location;
00138     string argument;
00139     for (int i = 0; i < count; i++) //for every command argument after the first (which is the
     filename)
00140     {
00141         argument = arg[i];
00142         if (argument[0] == '-') //argument prefix
00143         {
00144             if (argument[1] =='Z' || argument[1]== 'z') //zip code argument
00145             {
00146                 argument.erase(0,2); //erase the -z from the argument
00147                 zipcode = stoi(argument); //typecast the string zipcode into an integer
00148                 k.key = zipcode;
00149                 location = searchForPrimaryKey(k); //set the location of key if it exists,
00150                 k.byteLocation = location;
00151                 primaryKeyIndex.push_back(k); //push the key into the vector of primary keys
00152             }
00153         }
00154     }
00155     createPrimaryKeyIndexFile();
00156 }
00157 void buffer::createPrimaryKeyIndexFile(){
00158     ofstream outfile;
00159     outfile.open(myheader.indexFileName); //open the file
00160     primaryKey k;
00161
00162     outfile « "Key Locations marked as \"-1\" do not exist in the csv data file."«endl;
00163     outfile « string(70, '-') «endl;
00164     for (int i = 0; i < primaryKeyIndex.size(); i++)
00165     {
00166         k = primaryKeyIndex[i];
00167         outfile« left « setw(10)« "PrimaryKey: " « left « setw(6)«k.key « left« setw(1)« " | "« left «
```

```
      setw(10)«"Location: "« k.byteLocation « endl; //print key index to file
00168     }
00169     outfile.close(); //close the file
00170 }
00171 int buffer::searchForPrimaryKey(primaryKey k) { //searches to see if the primary key is in the file,
      if so, returns the location
00172
00173     string line;
00174     ifstream filein;
00175     filein.open("lengthIndicated.txt"); //open the file
00176     int location =0,totalLocation = 0;
00177     string numberString;
00178     getline(filein, line); //skip first line which contains the header information
00179
00180         for (int i = 0; i < myheader.recordCount; i++)
00181         {
00182             record r = vectorRecords[i];
00183             location = r.recordSize;
00184             if(stoi(r.zipcode) == k.key) //found the key and recorded location
00185             {
00186                 filein.close(); //close the file
00187                 return(totalLocation); //return the found key location
00188             }
00189             totalLocation = totalLocation + location; //update the total location
00190         }
00191             filein.close(); //close the file
00192             return(-1); //key was not found, set to negative number to indicate it wasn't
00193 }
00194 //see header for info
00195 void buffer::Write() {
00196
00197     sort(vectorRecords.begin(), vectorRecords.end()); //sort the vector to make processing easier
00198
00199 // Evaluate and ouput west,east,north, and south most zip codes for each state //
00200     record myrecord;
00201     myrecord = vectorRecords[0];
00202     string st = myrecord.state;
00203     const double reset = -999999; //value to restore westmost, eastmost, northmost, southmost values
      after every entry per state has been accounted for
00204     const int design = 55; //value for formatting output
00205
00206     double westMost = reset, eastMost = reset;
00207     double northMost = reset, southMost = reset; //stores the (west,east,north,south) most value for
      each state. Set to impossible value to start.
00208     string zipWestMost, zipEastMost, zipNorthMost, zipSouthMost; //stores the zip code for the
      (west,east,north,south) most values.
00209
00210     cout « string(design, '-') « endl; //header for output
00211     cout « setw(7) « "State"« setw(12)« "North-Most" « setw(12)« "South-Most" « setw(12)« "East-Most"
      « setw(12)« "West-Most" « endl; //header for output
00212     cout « string(design, '-')« endl; //header for output
00213
00214     for(int i = 0; i < vectorRecords.size(); i++) //for every record in the vector
00215     {
00216         myrecord = vectorRecords[i]; //set myrecord equal to current record being read
00217         if(st == myrecord.state){ //the same state is being read
00218             if(stod(myrecord.longitude) < westMost || westMost == reset) //check for west most
00219             {
00220                 westMost = stod(myrecord.longitude); //typecast the longitude and store it as the
      westmost value
00221                 zipWestMost = myrecord.zipcode; //store the zipcode for this record
00222             }
00223             if(stod(myrecord.longitude) > eastMost || eastMost == reset) //check for east most
00224             {
00225                 eastMost = stod(myrecord.longitude); //typecast the longitude and store it as the
      eastmost value
00226                 zipEastMost = myrecord.zipcode; //store the zipcode for this record
00227             }
00228             if(stod(myrecord.latitude) > northMost || northMost == reset) //check for south most
00229             {
00230                 northMost = stod(myrecord.latitude); //typecast the latitude and store it as the
      northmost value
00231                 zipNorthMost = myrecord.zipcode; //store the zipcode for this record
00232             }
00233             if(stod(myrecord.latitude) < southMost || southMost == reset) //check for north most
00234             {
00235                 southMost = stod(myrecord.latitude); //typecast the latitude and store it as the
      southmost value
00236                 zipSouthMost = myrecord.zipcode; //store the zipcode for this record
00237             }
00238         }
00239         else{
00240             cout « setw(7) « vectorRecords[i-1].state« setw(12)« zipNorthMost « setw(12)« zipSouthMost
      « setw(12)« zipEastMost « setw(12)« zipWestMost « endl; //output the info for the state
00241
00242             st = myrecord.state;
00243             westMost = reset; //reset to impossible value
```

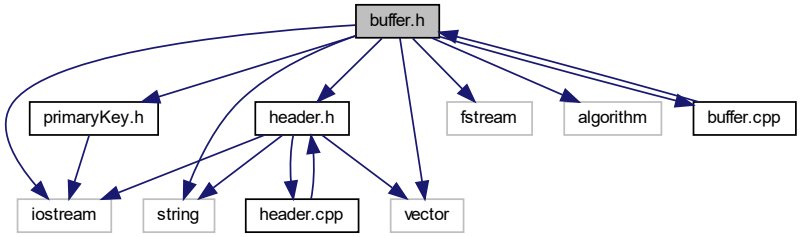
```
00244             eastMost = reset; //reset to impossible value
00245             northMost = reset; //reset to impossible value
00246             southMost = reset; //reset to impossible value
00247             i--; //decrement so that this current state is accounted for
00248         }
00249     }
00250      cout « string(design, '-')« endl; //footer for output
00251
00252 }
00253 void buffer:: PrintKeyData(){
00254     ifstream inFile;
00255     inFile.open("lengthIndicated.txt");
00256     string line;
00257     const int design = 140; //value for formatting output
00258     primaryKey k;
00259     bool falseKey = false;
00260     getline(inFile, line); //ignore the first line, which contains header information
00261
00262     cout « string(design, '-') « endl; //header for output
00263     cout « left« setw(12)« "Zipcode"« left«setw(40)« "City" « left«setw(12)« "State" « left«setw(40)«
      "County" « left«setw(15)« "Latitude" « left«setw(15)« "Longitude" « endl; //header for output
00264     cout « string(design, '-') « endl; //header for output
00265
00266     for (int i = 0; i < primaryKeyIndex.size(); i++) //for every primarykey in the index
00267     {
00268         k = primaryKeyIndex[i];
00269         if(k.byteLocation == -1){
00270             falseKey = true;
00271         }
00272         for (int i = 0; i < myheader.recordCount; i++)
00273         {
00274             record r = vectorRecords[i];
00275             if(stoi(r.zipcode) == k.key) //found the key and recorded location
00276             {
00277                 cout « left« setw(12)« r.zipcode« left«setw(40)« r.city« left«setw(12)« r.state«
    left«setw(40)« r.county « left«setw(15)« r.latitude« left«setw(15)« r.longitude « endl;
00278             }
00279         }
00280         inFile.close();
00281     }
00282     if (falseKey) //there was an incorrect key found, output all of the incorrect keys
00283     {
00284         cout « "\n\n\nThese keys did not exist in the file: ";
00285         for (int i = 0; i < primaryKeyIndex.size(); i++)
00286         {
00287             k = primaryKeyIndex[i];
00288             if(k.byteLocation == -1) //the location was not found for this key
00289             {
00290                 cout «" " «k.key « ","; //output the key
00291             }
00292         }
00293     }
00294 }
00295 bool operator <(const record& r1, const record& r2) {
00296     return r1.state < r2.state;  //compare the state of two records, to help with sorting
00297 }
```

4.3 buffer.h File Reference

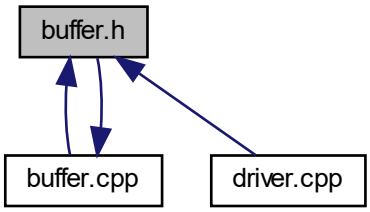
The header file for the class 'buffer'.

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>
#include "header.h"
#include "primaryKey.h"
#include "buffer.cpp"
```

Include dependency graph for buffer.h:



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



Classes

- class record

  class for the structure of data stored as strings, and the size of the record stored as an integer
- class buffer

  class that stores data into a vector and reads, writes, packs and unpacks data

4.3.1 Detailed Description

The header file for the class 'buffer'.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file buffer.h.

4.4 buffer.h

Go to the documentation of this file.

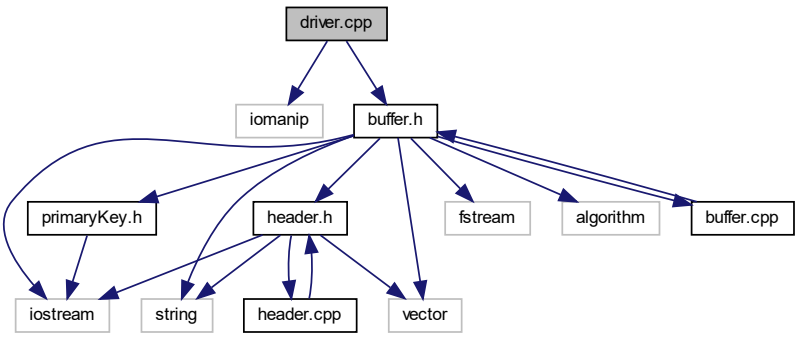
```
00001 //-------------------------------------------------------------------------
00005 //-------------------------------------------------------------------------
00006
00007 #ifndef BUFFER_H
00008 #define BUFFER_H
00009 #include <iostream>
00010 #include <string>
00011 #include <vector>
00012 #include <fstream>
00013 #include <algorithm>
00014 #include "header.h"
00015 #include "primaryKey.h"
00016 using namespace std;
00017
00023 class record {
00024
00025 public:
00026     string zipcode;
00027     string city;
00028     string state;
00029     string county;
00030     string latitude;
00031     string longitude;
00032     int recordSize;
00033 };
00034
00041 class buffer {
00042
00043 public:
00044
00045     vector <record> vectorRecords; //vector that will store records
00046     vector <primaryKey> primaryKeyIndex; //vector for storing primary keys
00047     header myheader; //header which contains information about the file
00048
00058     void EvaluateArguments(string arr[], int count);
00059
00069     void Read(string file);
00070
00079     int searchForPrimaryKey(primaryKey);
00080
00088     void createPrimaryKeyIndexFile();
00089
00097     void PrintKeyData();
00098
00106     void Write();
00107
00116     record unpack(string);
00117
00126     void pack(record);
00127
00135     void createLengthIndicatedFile(string);
00136     friend bool operator< (const record& r1, const record& r2); //overload < operator
00137 };
00138 #include "buffer.cpp"
00139 #endif
```

4.5 driver.cpp File Reference

This is the driver file.

```
#include <iomanip>
#include "buffer.h"
```

Include dependency graph for driver.cpp:



Functions

- int main (int argc, char ∗argv[ ])

4.5.1 Detailed Description

This is the driver file.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file driver.cpp.

4.5.2 Function Documentation

4.5.2.1 main()

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

Definition at line 11 of file driver.cpp.

```
00012 {
00013     if (argc == 1) //user did not enter in any arguments
00014     {
00015         cout « "You need to enter in arguments! Enter then in like so: ./a.exe filename.extention
      arguments" « endl;
00016         abort(); //abort to avoid errors
00017     }
00018
00019     string file = argv[1]; //first argument is the name of the csv file
00020     string arr[argc];
```

```
00021     for (int i = 2; i < argc; i++) //for every command argument after the first (which is the
      filename)
00022     {
00023         arr[i-2] = argv[i];
00024     }
00025     buffer mybuffer; //create buffer instance
00026     mybuffer.Read(file);
00027     if(argc > 2) //user entered at least one argument after the filename
00028     {
00029         mybuffer.EvaluateArguments(arr, argc);
00030         mybuffer.PrintKeyData();
00031     }
00032     else //user only entered the filename (outputs Zipcode 1.0 functionality)
00033     {
00034         mybuffer.Write();
00035     }
00036     return 0;
00037 }
```

4.6 driver.cpp

Go to the documentation of this file.

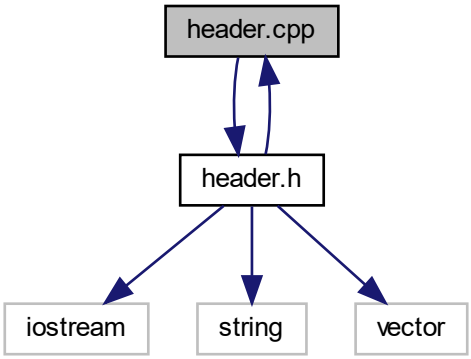
```
00001 //-----------------------------------------------------------------------------
00005 //-----------------------------------------------------------------------------
00006 #include <iomanip>
00007 #include "buffer.h"
00008
00009 using namespace std;
00010
00011 int main(int argc, char* argv[]) //take in arguments from command line
00012 {
00013     if (argc == 1) //user did not enter in any arguments
00014     {
00015         cout « "You need to enter in arguments! Enter then in like so: ./a.exe filename.extention
      arguments" « endl;
00016         abort(); //abort to avoid errors
00017     }
00018
00019     string file = argv[1]; //first argument is the name of the csv file
00020     string arr[argc];
00021     for (int i = 2; i < argc; i++) //for every command argument after the first (which is the
      filename)
00022     {
00023         arr[i-2] = argv[i];
00024     }
00025     buffer mybuffer; //create buffer instance
00026     mybuffer.Read(file);
00027     if(argc > 2) //user entered at least one argument after the filename
00028     {
00029         mybuffer.EvaluateArguments(arr, argc);
00030         mybuffer.PrintKeyData();
00031     }
00032     else //user only entered the filename (outputs Zipcode 1.0 functionality)
00033     {
00034         mybuffer.Write();
00035     }
00036     return 0;
00037 }
```

4.7 header.cpp File Reference

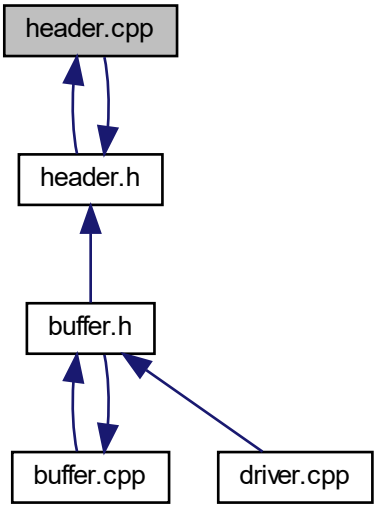
The file containing the header information.

```
#include "header.h"
```

Include dependency graph for header.cpp:



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



4.7.1 Detailed Description

The file containing the header information.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file header.cpp.

4.8 header.cpp

Go to the documentation of this file.

```
00001 //---------------------------------------------------------------------------
00005 //---------------------------------------------------------------------------
00006 #include "header.h"
00007
00008 string header::addHeaderInformation()
00009 {
00010     string hInfo; //stores all header information, returns at the end of function.
00011
00012     structureType = "Length Indicated Records, Comma Seperated fields";
00013     version = 2.0;
00014     // headerSize determined after seeing length of header
00015     // recordSizeByte hard coded in header.h
00016     recordSizeFormat = "ASCII";
00017     indexFileName = "indexFile.txt";
00018     indexSchema = "Zipcode, ByteLocation";
00019     //fieldCount hard coded in header.h
00020     //record cound set in buffer::Read()
00021     //primaryKey set in determineOrder
00022     //fieldOrder set in determineOrder
00023
00024     hInfo = "Structure: " + structureType + ","+
00025             "Version: " + to_string(version) + ","+
00026             "Record Size Byte: " + to_string(recordSizeByte) + ","+
00027             "Record Size Format: " + recordSizeFormat + ","+
00028             "Index File Name: " + indexFileName + ","+
00029             "Index Schema: " + indexSchema + ","+
00030             "Number of fields in each record: " + to_string(fieldCount) + ","+
00031             "Number of records in file: " + to_string(recordCount) + ","+
00032             "Primary Key: " + to_string(primaryKey) + ","+
00033             "Field Order: [" + to_string(fieldOrder[0]) + ","+ to_string(fieldOrder[1]) + ","+
      to_string(fieldOrder[2]) + ","+ to_string(fieldOrder[3]) + ","+ to_string(fieldOrder[4]) + ","+
      to_string(fieldOrder[5]) + "]\n";
00034
00035     headerSize = hInfo.size();
00036     string headerSizeString = to_string(headerSize);
00037     headerSize = headerSize + headerSizeString.length(); //set headerSize equal to the contents of the
      header, the number of bytes the headerSize takes, and + 1 for the comma included after the size.
00038
00039     hInfo.insert(0, to_string(headerSize) + ","); //insert the headersize to the beginning of the
      header
00040     return hInfo; //return the header string
00041 }
00042
00043 void header::determineOrder(string columnHeaders)
00044 {
00045     string label ="";
00046     int j = 0;
00047     for (int i = 0; i < columnHeaders.size(); i++) //for every character in the column header
00048     {
00049         if(columnHeaders[i] != ',') //read in characters until it forms a readable label
00050         {
00051             label = label + columnHeaders[i];
00052         }
00053         if(label == "\"ZipCode\""){
00054             fieldOrder[j] = 1;
00055             primaryKey = j+1;
00056             j++;
00057             label = ""; //reset label
00058         }
00059         else if(label == "\"PlaceName\""){
00060             fieldOrder[j] = 2;
00061             j++;
00062             label = ""; //reset label
00063         }
00064         else if(label == "State"){
00065             fieldOrder[j] = 3;
00066             j++;
00067             label = ""; //reset label
00068         }
00069         else if(label == "County"){
00070             fieldOrder[j] = 4;
00071             j++;
00072             label = ""; //reset label
00073         }
00074         else if(label == "Lat"){
00075             fieldOrder[j] = 5;
00076             j++;
00077             label = ""; //reset label
00078         }
00079         else if(label == "Long"){
00080             fieldOrder[j] = 6;
00081             j++;
```

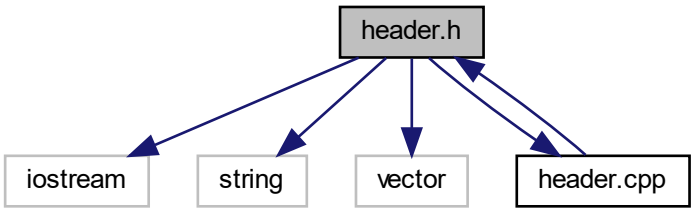
```
00082             label = ""; //reset label
00083         }
00084     }
00085 }
```

4.9 header.h File Reference

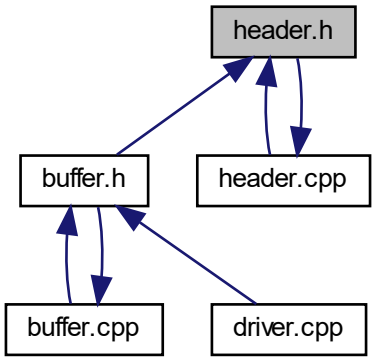
The header file for the class 'header'.

```
#include <iostream>
#include <string>
#include <vector>
#include "header.cpp"
```

Include dependency graph for header.h:



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



Classes

- class header

  class for the structure of the header

4.9.1 Detailed Description

The header file for the class 'header'.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file header.h.

4.10 header.h

Go to the documentation of this file.

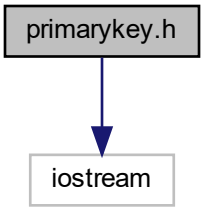
```
00001 //-----------------------------------------------------------------------------
00005 //-----------------------------------------------------------------------------
00006 #ifndef HEADER_H
00007 #define HEADER_H
00008 #include <iostream>
00009 #include <string>
00010 #include <vector>
00011 using namespace std;
00012
00018 class header{
00019 public:
00020
00027     string addHeaderInformation(); //adds the header file information from the file and then returns
       the string with the header information added
00028
00037     void determineOrder(string); //determines the order of the fields in the record, and updates
       fieldOrder[]
00038
00039     int headerSize; //size of the header
00040     string structureType; //type of record (Length indicated, comma seperated, etc.)
00041     double version; //version # of the file
00042     int recordSizeByte = 2; //number of bytes that indicates the size of each record
00043     string recordSizeFormat; //How the record size's number is formatted (ASCII)
00044     string indexFileName; //Name of the index file
00045     string indexSchema; //What the index file is organized like (Zipcode, Reference #) (example:
       56303,47291)
00046     int recordCount; //Number of records in file
00047     int fieldCount = 6; //Number of fields per record (6 including zipcode,city,state,county,lat,lon)
00048     int primaryKey; //Which field is the primary key
00049     int fieldOrder[]; //keeps track of the order of the fields in a record (in cases where the csv is
       column randomized)
00050
00051 };
00052 #include "header.cpp"
00053 #endif
```

4.11 primarykey.h File Reference

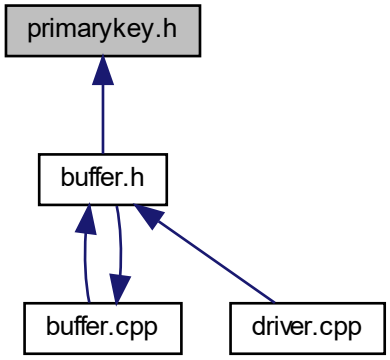
The header file for the class 'primaryKey'.

```
#include <iostream>
```

Include dependency graph for primarykey.h:



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



Classes

- class primaryKey

  class for the structure of the header

4.11.1 Detailed Description

The header file for the class 'primaryKey'.

Author

Evan Burdick, Joseph Kuzko,& Matthew Xiong

Definition in file primarykey.h.

4.12 primarykey.h

Go to the documentation of this file.

```
00001 //-------------------------------------------------------------------------
00005 //-------------------------------------------------------------------------
00006 #ifndef PRIMARYKEY_H
00007 #define PRIMARYKEY_H
00008 #include <iostream>
00009 using namespace std;
00010
00016 class primaryKey{
00017 public:
00018     int key;
00019     int byteLocation;
00020 };
00021 #endif
```

Index