Project 5 - Zipcodes Version 3

Generated by Doxygen 1.9.5

Chapter 1

readme

Project 4 Instructions

Compiling Compile with g++ –std=c++17 -Wall -o ZipCode.out ∗.cpp -lstdc++fs

(-lstdc++fs links the filesystem library and must be at the end of the call to g++)

Running There are two optional arguments that will change the way the program functions.

-Z<list_of_zip_codes> where list_of_zip_codes is a comma separated list of zip codes to search the file for

-C<csv_filename> where csv_filename is the name of a csv file of the type used in this project.

The only required argument is a path to a file.

If -C is used, the program will try to convert the csv file to the lirf format and write it to the lirf path provided.
If -Z is used, the program will load the file from the path provided and search for the zip codes.
If both -C and -Z are used, the program will currently act as if only -C was used. (This might change later)

These options can appear in any order after the executable name.

Examples:

./ZipCode.out -Z24321,42444 example_file
will try to open and read a file called example_file and search for the zip codes 24321 and 42444. Any zip codes that are found in the file will have their entire record printed in a table. The zip codes that are not found will be listed after any that were found.

1.0.1 For example, using these two zip codes on the file provided, the output will look like

1.0.2 Zip Place NameState County Latitude Longitude

1.0.3 42444 Poole KY Webster 37.641 -87.6439

The following zip codes did not match any records in the file: 24321

./ZipCode.out -Cexample_csv_file.csv example_file
will try to open and read a file called example_csv_file.csv and will create or overwrite a file called example_file with the data read from the csv file.

./ZipCode.out example_file
without either optional argument, the program will simply try to read example_file as a file and do nothing.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 BlockBuffer Class Reference

Public Member Functions

- void processBlockHeader ()
- void setBlockHeader (BlockHeader blockHeader)
- BlockBuffer ()=default
- BlockBuffer (BlockFileHeader &header)
- bool read (istream &file, int RBN)
- void write (ostream &file, int RBN)
- bool unpack (LengthIndicatedBuffer< BlockFileHeader > &lBuf)
- void pack (PlaceBlock pb, LengthIndicatedBuffer< BlockFileHeader > &lBuf)
- void pack (LengthIndicatedBuffer< BlockFileHeader > &lBuf)
- void setPrecedingBlock (int RBN)
- void setSucceedingBlock (int RBN)
- void clear ()
- void **init** (BlockFileHeader &header)

Public Attributes

- std::vector< char > **buffer**
- BlockFileHeader header
- BlockHeader blockHeader
- int unpackedRecords = 0
- int curr = 0
- int length = 0
- int RBN

Private Member Functions

- bool read (istream &file)

4.1.1 Constructor & Destructor Documentation

4.1.1.1 BlockBuffer() [1/2]

```
BlockBuffer::BlockBuffer ( )  [default]
```

default contructor

Precondition

none

Postcondition

class comes as defined

4.1.1.2 BlockBuffer() [2/2]

```
BlockBuffer::BlockBuffer (
            BlockFileHeader & header )
```

constructor that specifies a header for the file

Parameters

header	the header to set in-class

Precondition

none

Postcondition

class has custom file header

4.1.2 Member Function Documentation

4.1.2.1 clear()

```
void BlockBuffer::clear ( )
```

resets the buffer

Precondition

none

Postcondition

empties the buffer and sets variables accordingly

Here is the caller graph for this function:

4.1.2.2 pack() [1/2]

```
void BlockBuffer::pack (
            LengthIndicatedBuffer< BlockFileHeader > & lBuf )
```

packs data into the block buffer

Parameters

lBuf	length buffer where data is retrieved

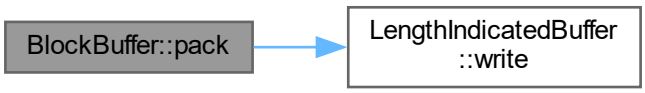
Precondition

none

Postcondition

data is packed into block buffer

Here is the call graph for this function:



4.1.2.3 pack() [2/2]

```
void BlockBuffer::pack (
            PlaceBlock pb,
            LengthIndicatedBuffer< BlockFileHeader > & lBuf )
```

packs data into block buffer for entire block

Parameters

pb	place block class used to find all lines of block
lBuf	length buffer that gets data taken from it

Precondition

none

Postcondition

data is packed into block buffer

Here is the call graph for this function:

Here is the caller graph for this function:

4.1.2.4 processBlockHeader()

```
void BlockBuffer::processBlockHeader ( )
```

keeps track of the record block number processes the block header

Precondition

buffer contains block of records

Postcondition

places block header at beginning of block

4.1.2.5 read() [1/2]

```
bool BlockBuffer::read (
            istream & file )  [private]
```

reads the data in the file

Parameters

file	file to be read

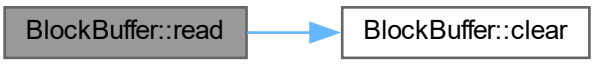
Precondition

none

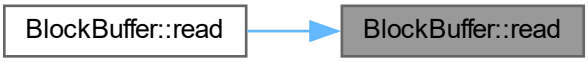
Postcondition

reads data into buffer. returns true or false based on if file was opened and read or not

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.6 read() [2/2]

```
bool BlockBuffer::read (
            istream & file,
            int RBN )
```

function that reads a particular record block number

Parameters

file	the file to be read
RBN	the record block number to get

Precondition

none

Postcondition

returns success or failure

Here is the call graph for this function:

4.1.2.7 setBlockHeader()

```
void BlockBuffer::setBlockHeader (
            BlockHeader blockHeader )
```

function used to determine contents of block header

Parameters

blockHeader	block header to be used

Precondition

none

Postcondition

sets in-class block header to equal block header placced into function

4.1.2.8 setPrecedingBlock()

```
void BlockBuffer::setPrecedingBlock (
            int RBN )
```

sets previous block number to the argument RBN

Parameters

RBN	block number of previous block to be set

Precondition

none

Postcondition

sets preceding block to the value of RBN

4.1.2.9 setSucceedingBlock()

```
void BlockBuffer::setSucceedingBlock (
            int RBN )
```

sets next block number to the argument RBN

Parameters

RBN	block number of next block to be set

Precondition

none

Postcondition

sets succeeding block to the value of RBN

4.1.2.10 unpack()

```
bool BlockBuffer::unpack (
            LengthIndicatedBuffer< BlockFileHeader > & lBuf )
```

unpacking buffer data into a length buffer

Parameters

lBuf	the length buffer

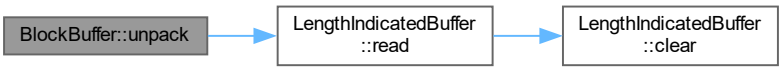
Precondition

data must be in the block buffer

Postcondition

data is read into the length buffer and returns true or false based on success

Here is the call graph for this function:



4.1.2.11 write()

```
void BlockBuffer::write (
            ostream & file,
            int RBN )
```

function that writes to a particular block

Parameters

file	the file to write to
RBN	desired record block number to write to

Precondition

none

Postcondition

writes data in desired file and block

4.1.3 Member Data Documentation

4.1.3.1 blockHeader

```
BlockHeader BlockBuffer::blockHeader
```

file header

4.1.3.2 curr

```
int BlockBuffer::curr = 0
```

records in buffer that remain unpacked

4.1.3.3 header

```
BlockFileHeader BlockBuffer::header
```

buffer itself

4.1.3.4 length

```
int BlockBuffer::length = 0
```

current position in buffer

4.1.3.5 RBN

```
int BlockBuffer::RBN
```

length of content in buffer

4.1.3.6 unpackedRecords

```
int BlockBuffer::unpackedRecords = 0
```

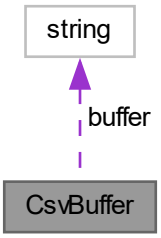
header for the block

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

- blockBuffer.h
- blockBuffer.cpp

4.2 CsvBuffer Class Reference

Collaboration diagram for CsvBuffer:



Public Member Functions

- CsvBuffer (const char delim=',')

  Construct a new Csv Buffer object.
- bool read (std::istream &instream)

  Reads one record into the buffer if there is data left in the stream.
- bool unpack (std::string &str)

  Reads a field and puts it into a string.
- void init (std::istream &instream)

  Performs the first read and extracts the headers.
- std::pair< HeaderField, std::string > getCurFieldHeader ()

  Gets the type and value of the current field.
- std::vector< std::pair< HeaderField, std::string > > **getHeaders** () const

Private Member Functions

- void readHeader ()

  Reads through the first record in the file and extracts the fields for use as metadata.

Private Attributes

- const char **delim**
- std::string **buffer**
- size_t **curr**

  holds the position of the start of unprocessed fields
- size_t fieldNum

  number of records currently in the unprocessed part of the buffer
- size_t **numFields**

  number of fields in each record
- std::vector< std::pair< HeaderField, std::string > > **headers**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 CsvBuffer()

```
CsvBuffer::CsvBuffer (
            const char delim = ',' )
```

Construct a new Csv Buffer object.

Parameters

delim	The delimiter used in the csv file

4.2.2 Member Function Documentation

4.2.2.1 getCurFieldHeader()

```
std::pair< HeaderField, std::string > CsvBuffer::getCurFieldHeader ( )
```

Gets the type and value of the current field.

Precondition

headers has been initialized

Postcondition

returns a pair containing the HeaderField type and the string value of the current field's header

Returns

std::pair<HeaderField, std::string>

Here is the caller graph for this function:

4.2.2.2 init()

```
void CsvBuffer::init (
            std::istream & instream )
```

Performs the first read and extracts the headers.

Parameters

in	*instream*	stream to be read from

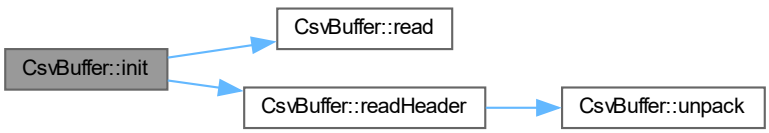
Precondition

buffer is empty

Postcondition

headers contains the values returned by readHeader
buffer contains one unprocessed record
curr points to the start of the record.
fieldNum is increased by one if the record contains more fields or is set to zero if the entire record has been read.

Here is the call graph for this function:



4.2.2.3 read()

```
bool CsvBuffer::read (
            std::istream & instream )
```

Reads one record into the buffer if there is data left in the stream.

Parameters

in	*instream*	one record will be read

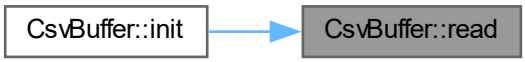
Precondition

instream is an open stream that contains data in a CSV format

Postcondition

buffer contains data to be unpacked
instream points to next record or end of stream

Here is the caller graph for this function:



4.2.2.4 readHeader()

```
void CsvBuffer::readHeader ( )  [private]
```

Reads through the first record in the file and extracts the fields for use as metadata.

Precondition

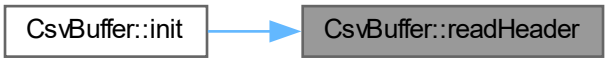
headers is an empty vector and the buffer has had data read into it

Postcondition

headers contains the pairs of header field types and the values of the header fields
curr points to the first record
numFields is equal to number of header fields found

Here is the call graph for this function:

Here is the caller graph for this function:



4.2.2.5 unpack()

```
bool CsvBuffer::unpack (
            std::string & str )
```

Reads a field and puts it into a string.

Parameters

out	*str*	the string that will hold the value of the field

Returns

true record has not had every field unpacked

false record has no more fields to unpack

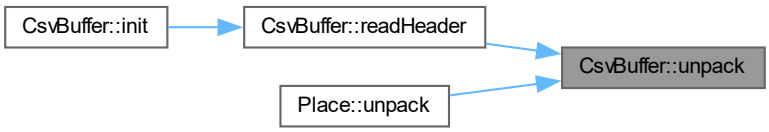
Precondition

curr is pointing to the start of a field str is an empty std::string

Postcondition

str contains the value of the field curr is pointing to the start of the next field or the end of the record

Here is the caller graph for this function:



4.2.3 Member Data Documentation

4.2.3.1 fieldNum

```
size_t CsvBuffer::fieldNum  [private]
```

number of records currently in the unprocessed part of the buffer

keeps track of how many fields of a record have been processed

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

- CsvBuffer.h
- CsvBuffer.cpp

4.3 FieldInfo Struct Reference

Public Attributes

- char **fieldName** [50]
- HeaderField **fieldType**

  the name of the field

Friends

- std::istream & **operator**>> (std::istream &ins, FieldInfo &fieldInfo)

  the HeaderField type of the field
- std::ostream & **operator**<< (std::ostream &os, FieldInfo &fieldInfo)

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

- Header.h

4.4 FileInfo Struct Reference

Public Attributes

- int **lengthIndicatorSize**
- LengthIndicatorType **lengthIndicatorFormat**

  number of bytes in length indicator
- int **fieldsPerRecord**

  ASCII, BINARY, or BCD.
- int **primaryKeyPosition**

  number of fields in each record
- char **indexFileName** [100]

  the ordinal position of the primary key used to index the file

Friends

- std::istream & **operator**>> (std::istream &ins, FileInfo &fileInfo)

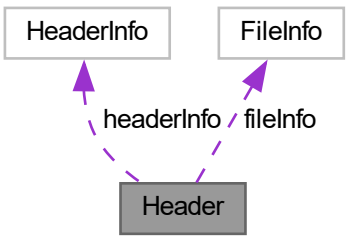
  the name of the index file to be loaded at program start
- std::ostream & **operator**<< (std::ostream &os, FileInfo &fileInfo)

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

- Header.h

4.5 Header Struct Reference

Collaboration diagram for Header:



Public Attributes

- HeaderInfo **headerInfo**
- FileInfo **fileInfo**
- std::vector< FieldInfo > **fields**

Friends

- std::ostream & **operator**<< (std::ostream &os, Header &header)

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

- Header.h

4.6 HeaderBuffer Class Reference

Public Member Functions

- void read (std::istream &ins)

  Read header file into buffer.
- Header unpack ()

  Unpacks header fields from buffer into Header object.

Private Attributes

- std::vector< unsigned char > **buffer**

4.6.1 Member Function Documentation

4.6.1.1 read()

```
void HeaderBuffer::read (
            std::istream & ins )
```

Read header file into buffer.

Parameters

in	*ins*	stream to be read in from

Precondition

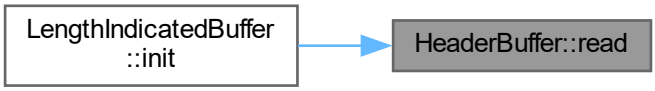
ins is pointing to an open length indicated file and ins.good() is true

Postcondition

ins is positioned at the first character after the header

buffer is filled with header bytes

Here is the caller graph for this function:



4.6.1.2 unpack()

```
Header HeaderBuffer::unpack ( )
```

Unpacks header fields from buffer into Header object.

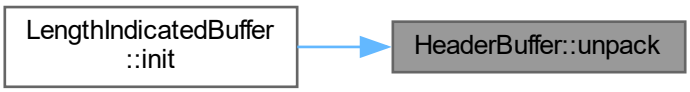
Precondition

read() has been called on a valid file and buffer contains a header with the Header format

Returns

a Header object loaded with the information in the file

Here is the caller graph for this function:



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

- HeaderBuffer.h
- HeaderBuffer.cpp

4.7 HeaderInfo Struct Reference

Public Attributes

- char **magic** [4]
- int **version**

  4 bytes at the start indicating that the file is of the correct type
- int **headerSize**

  version number

Friends

- std::istream & **operator**>> (std::istream &ins, HeaderInfo &headerInfo)

  size of header in bytes, including header info
- std::ostream & **operator**<< (std::ostream &os, HeaderInfo &headerInfo)

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

- Header.h

4.8 PrimaryKey::IndexFileHeader Struct Reference

Public Attributes

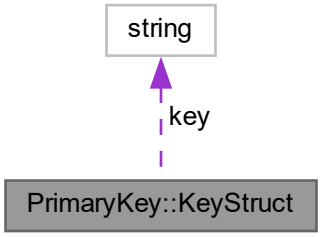
- int **version**
- int **keyCount**
- int **format**

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

- PrimaryKey.h

4.9 PrimaryKey::KeyStruct Struct Reference

Collaboration diagram for PrimaryKey::KeyStruct:



Public Attributes

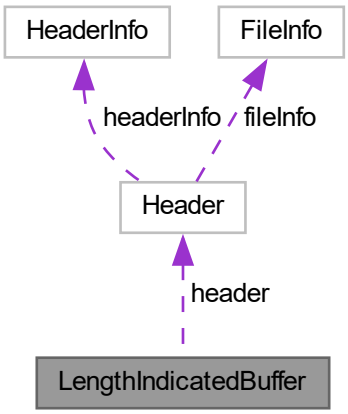
- std::string **key**
- unsigned int **offset**

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

- PrimaryKey.h

4.10 LengthIndicatedBuffer Class Reference

Collaboration diagram for LengthIndicatedBuffer:



Public Member Functions

- LengthIndicatedBuffer (const char delim=',')

  Construct a new LengthIndicatedBuffer object.
- bool read (std::istream &instream)

  Read a single length indicated record into the buffer.
- bool read (std::istream &instream, int indexOffset)

  Seek to the specified offset in the file and read a single length indicated record into the buffer.
- bool unpack (std::string &str)

  Reads a field and puts it into a string.
- void pack (const std::string str)

  Packs a field into the buffer.
- void write (std::ostream &outstream)

  Writes length of the field and the data in the buffer to the stream.
- bool init (std::istream &instream)

  read and extract the header.

- void writeHeader (std::ostream &outstream)

  Seeks to the start of the stream and writes the header member to the stream.
- void clear ()

  Sets curr to start of buffer.
- bool checkFileType (std::istream &instream)

  read the first 4 bytes of the file and check against the magic number.
- std::string getIndexFileName ()

  get the name of the index file from the header
- FieldInfo getCurFieldHeader ()

  Gets the type and value of the current field.

Public Attributes

- Header **header**

Private Attributes

- const char **delim**
- int **recordLength**
- bool **initialized** = false
- char **buffer** [1000]
- int **curr**

  holds the position of the start of unprocessed fields
- size_t **fieldNum** = 0

  keeps track of how many fields of a record have been processed
- size_t **numFields**

  number of fields in each record

4.10.1 Constructor & Destructor Documentation

4.10.1.1 LengthIndicatedBuffer()

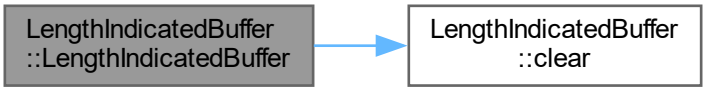
```
LengthIndicatedBuffer::LengthIndicatedBuffer (
            const char delim = ',' )
```

Construct a new LengthIndicatedBuffer object.

Parameters

delim	The delimiter used between the record fields

Here is the call graph for this function:



4.10.2 Member Function Documentation

4.10.2.1 checkFileType()

```
bool LengthIndicatedBuffer::checkFileType (
            std::istream & instream )
```

read the first 4 bytes of the file and check against the magic number.

Parameters

in	*instream*	stream to be read from

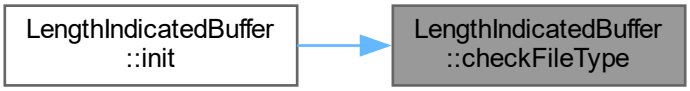
Precondition

instream is open for reading

Return values

true	if file has correct magic number
false	if file does not have correct magic number

Here is the caller graph for this function:



4.10.2.2 clear()

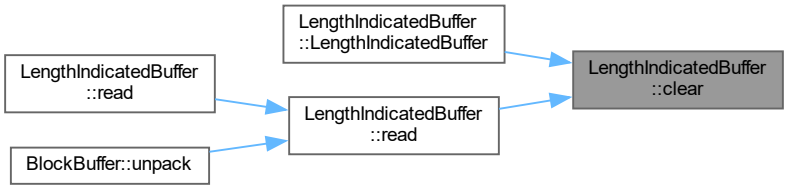
```
void LengthIndicatedBuffer::clear ( )
```

Sets curr to start of buffer.

Postcondition

curr = 0

Here is the caller graph for this function:



4.10.2.3 getCurFieldHeader()

```
FieldInfo LengthIndicatedBuffer::getCurFieldHeader ( )
```

Gets the type and value of the current field.

Precondition

headers has been initialized

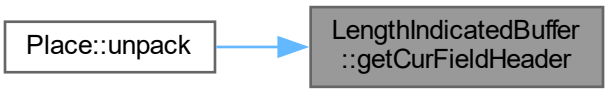
Postcondition

returns a FieldInfo struct with the field name and field type

Returns

FieldInfo

Here is the caller graph for this function:



4.10.2.4 getIndexFileName()

```
std::string LengthIndicatedBuffer::getIndexFileName ( )
```

get the name of the index file from the header

Precondition

initialized = true

Returns

string containing index file name

4.10.2.5 init()

```
bool LengthIndicatedBuffer::init (
            std::istream & instream )
```

read and extract the header.

Parameters

in	*instream*	stream to be read from

Precondition

instream points to a valid length indicated file opened for reading

Postcondition

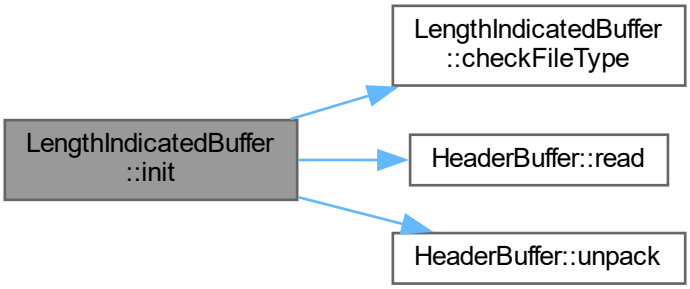
header has been loaded with the values from the stream

initialized has been set to true if header read was successful, false if it was not

Return values

true	header read was successful
false	header read was not successful

Here is the call graph for this function:



4.10.2.6 pack()

```
void LengthIndicatedBuffer::pack (
            const std::string str )
```

Packs a field into the buffer.

Parameters

in	*str*	the string that will holds the value of the field

Precondition

str is of the correct type indicated by headers[fieldNum].fieldType

Postcondition

if the field is not the first, buffer has had a comma and the data from str minus the null terminator added

if the field is first, buffer has had the data from str minus the null terminator added

curr points to the first position in buffer after the newly added field

4.10.2.7 read() [1/2]

```
bool LengthIndicatedBuffer::read (
            std::istream & instream )
```

Read a single length indicated record into the buffer.

Parameters

in	*instream*	the stream that points to the record

Precondition

instream is an open stream pointing to the start of a length indicated record

record size $<$ sizeof(buffer)

Postcondition

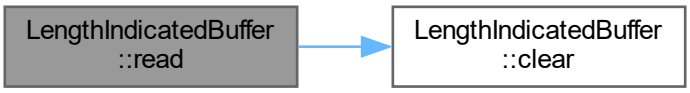
instream is positioned at the start of the next record or the end of the stream

curr is set to 0 and buffer is zeroed

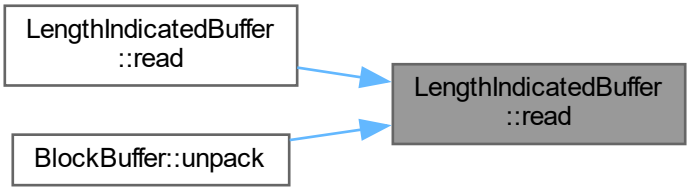
Return values

true	when instream.good() is true (indicating that we can read another record)
false	when instream.good() is false (indicating that we are probably at the end of the file)

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.2.8 read() [2/2]

```
bool LengthIndicatedBuffer::read (
            std::istream & instream,
            int indexOffset )
```

Seek to the specified offset in the file and read a single length indicated record into the buffer.

Parameters

in	*instream*	the stream that points to the record
in	*indexOffset*	the number of bytes from the start of the file to seek to before reading

Precondition

instream is an open stream pointing to the start of a length indicated record

indexOffset is the number of bytes from the start of the file to the start of a valid length indicated record

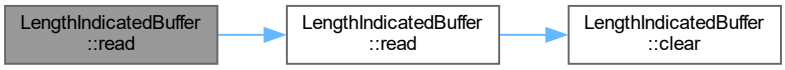
Postcondition

instream is positioned at the start of the next record or the end of the stream

Return values

true	when instream.good() is true (indicating that we can read another record)
false	when instream.good() is false (indicating that we are probably at the end of the file)

Here is the call graph for this function:



4.10.2.9 unpack()

```
bool LengthIndicatedBuffer::unpack (
            std::string & str )
```

Reads a field and puts it into a string.

Parameters

out	*str*	the string that will hold the value of the field

Precondition

curr is pointing to the start of a field str is an empty std::string

Postcondition

str contains the value of the field

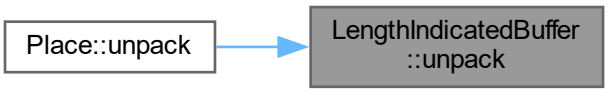
curr is pointing to the start of the next field or the next record

fieldNum is pointing to the type of the next field

Return values

true	record has not had every field unpacked
false	record has no more fields to unpack

Here is the caller graph for this function:



4.10.2.10 write()

```
void LengthIndicatedBuffer::write (
            std::ostream & outstream )
```

Writes length of the field and the data in the buffer to the stream.

Parameters

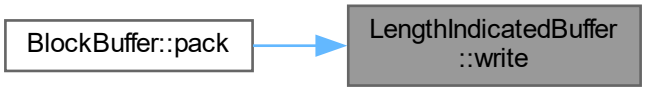
in	*str*	the string that will holds the value of the field

Precondition

str is of the correct type indicated by headers[fieldNum].fieldType

Postcondition

Here is the caller graph for this function:



4.10.2.11 writeHeader()

```
void LengthIndicatedBuffer::writeHeader (
            std::ostream & outstream )
```

Seeks to the start of the stream and writes the header member to the stream.

Postcondition

outstream is pointing to the first byte after the header

Parameters

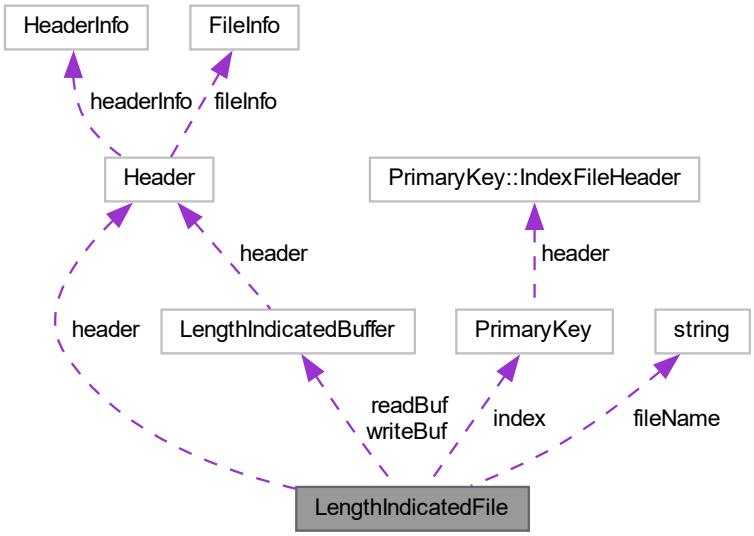
out	*outstream*	the stream to be written to

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

- LengthIndicatedBuffer.h
- LengthIndicatedBuffer.cpp

4.11 LengthIndicatedFile Class Reference

Collaboration diagram for LengthIndicatedFile:



Public Member Functions

- **LengthIndicatedFile** (std::string fileName)
- void **initializeBuffers** ()
- void **initializeIndex** ()
- bool **openDataFile** ()
- std::optional< Place > **findRecord** (std::string recordKey)
- void **generateIndex** ()
- bool **indexFileExists** ()

Private Attributes

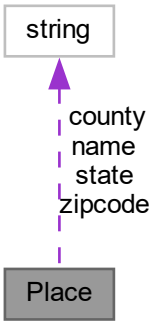
- Header **header**
- PrimaryKey **index**
- LengthIndicatedBuffer **readBuf**
- LengthIndicatedBuffer **writeBuf**
- std::string **fileName**
- std::fstream **file**
- int **dataStart**

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

- LengthIndicatedFile.h
- LengthIndicatedFile.cpp

4.12 Place Class Reference

Collaboration diagram for Place:



Public Member Functions

- Place (const Place &loc)
  Copy constructor.
- std::string getZipCode () const
  Returns the Zip Code.
- std::string getState () const
  Returns the 2 digit State Id.
- std::string getName () const
  Returns the Place Name.
- std::string getCounty () const
  Returns the County.
- double getLat () const
  Returns the latitude.
- double getLongi () const
  Returns the longitude.
- void unpack (CsvBuffer &buffer)
  Reads a record from the buffer and unpacks the fields into the class members.
- void unpack (LengthIndicatedBuffer &buffer)
  Reads a record from the buffer and unpacks the fields into the class members.
- void **pack** (LengthIndicatedBuffer &buffer)
- void operator= (const Place &loc)
  Assignment operator overload.
- size_t getSize ()
  get size of object in bytes
- void **print** ()

Private Attributes

- std::string **zipcode**
- std::string **name**
- std::string **state**
- std::string **county**
- double **latitude**
- double **longitude**

4.12.1 Constructor & Destructor Documentation

4.12.1.1 Place()

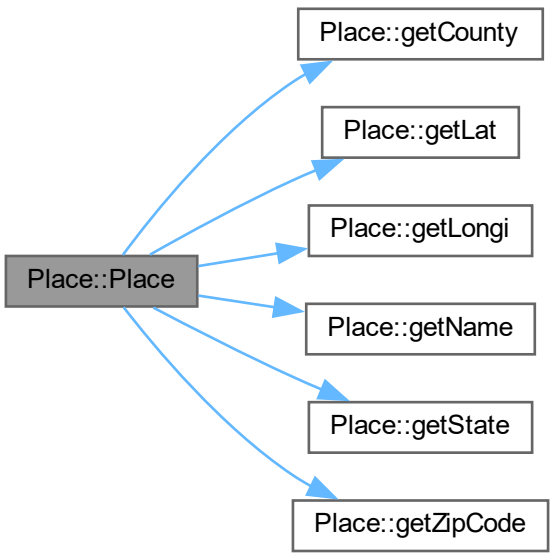
```
Place::Place (
            const Place & loc )
```

Copy constructor.

Parameters

loc	

Here is the call graph for this function:



4.12.2 Member Function Documentation

4.12.2.1 getCounty()

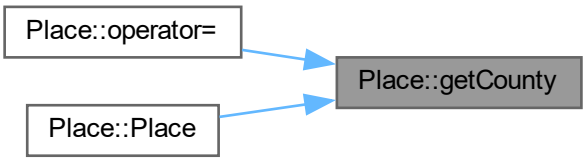
```
string Place::getCounty ( ) const
```

Returns the County.

Returns

std::string

Here is the caller graph for this function:



4.12.2.2 getLat()

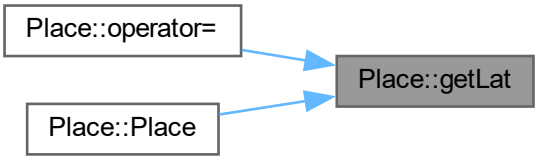
```
double Place::getLat ( ) const
```

Returns the latitude.

Returns

double

Here is the caller graph for this function:



4.12.2.3 getLongi()

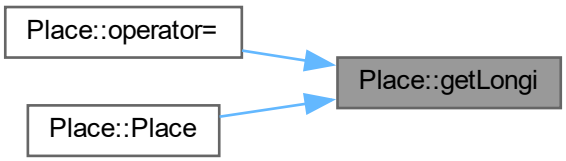
```
double Place::getLongi ( ) const
```

Returns the longitude.

Returns

double

Here is the caller graph for this function:



4.12.2.4 getName()

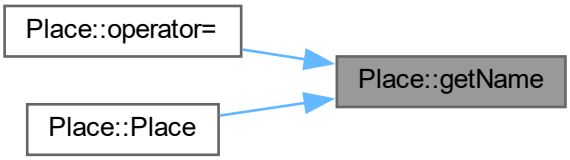
```
string Place::getName ( ) const
```

Returns the Place Name.

Returns

std::string

Here is the caller graph for this function:



4.12.2.5 getSize()

```
size_t Place::getSize ( )
```

get size of object in bytes

Returns

size_t size of object

4.12.2.6 getState()

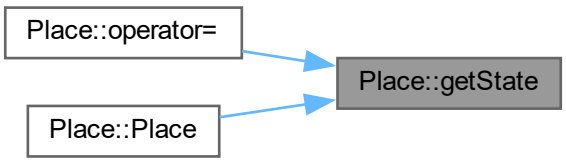
```
string Place::getState ( ) const
```

Returns the 2 digit State Id.

Returns

std::string

Here is the caller graph for this function:



4.12.2.7 getZipCode()

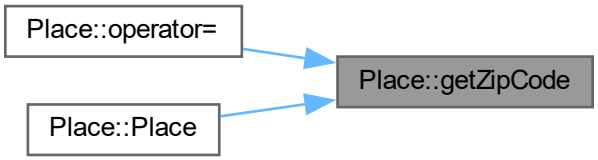
```
string Place::getZipCode ( ) const
```

Returns the Zip Code.

Returns

std::string

Here is the caller graph for this function:



4.12.2.8 operator=()

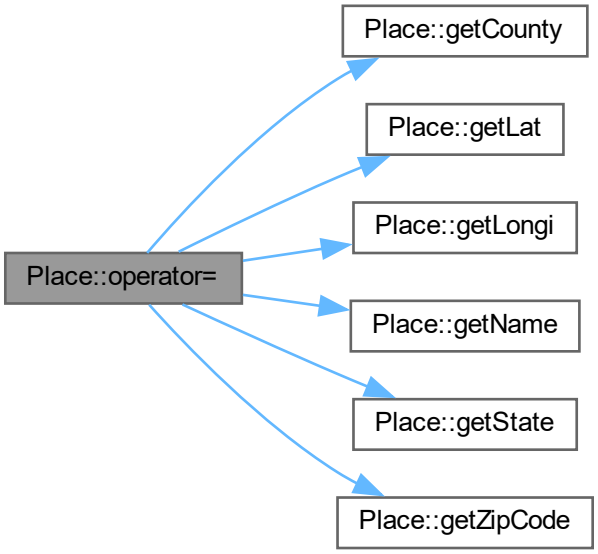
```
void Place::operator= (
            const Place & loc )
```

Assignment operator overload.

Parameters

loc	The place object that this one's parameters will match

Here is the call graph for this function:



4.12.2.9 unpack() [1/2]

```
void Place::unpack (
            CsvBuffer & buffer )
```

Reads a record from the buffer and unpacks the fields into the class members.

Parameters

in,out	*buffer*	The buffer to be read from

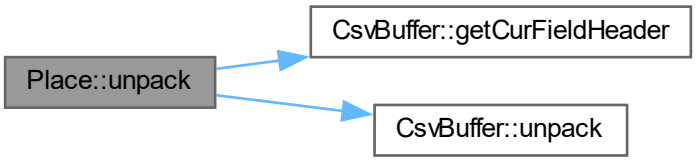
Precondition

buffer has a record that contains zipcode, place name, state id, county, latitude, and longitude fields

Postcondition

the member variables have been set to the values mentioned above, if the column with that name was found

Here is the call graph for this function:



4.12.2.10 unpack() [2/2]

```
void Place::unpack (
            LengthIndicatedBuffer & buffer )
```

Reads a record from the buffer and unpacks the fields into the class members.

Parameters

in,out	*buffer*	The buffer to be read from

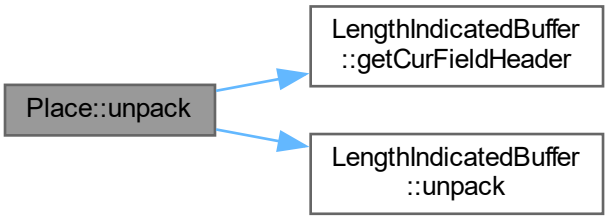
Precondition

buffer has a record that contains zipcode, place name, state id, county, latitude, and longitude fields

Postcondition

the member variables have been set to the values mentioned above, if the column with that name was found

Here is the call graph for this function:

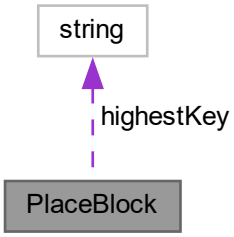


The documentation for this class was generated from the following files:

- Place.h
- Place.cpp

4.13 PlaceBlock Class Reference

Collaboration diagram for PlaceBlock:



Public Member Functions

- PlaceBlock ()=default
- bool **unpack** (BlockBuffer &bBuf, LengthIndicatedBuffer< BlockFileHeader > &lBuf)
- string **getHighestKey** ()
- optional< Place > getRecord (const string &key)
- void print ()
- void sort ()
- int getSize ()

Public Attributes

- BlockHeader blockHeader
- vector< Place > placeBlock

Private Attributes

- int **size** = -1
- string highestKey = ""

4.13.1 Constructor & Destructor Documentation

4.13.1.1 PlaceBlock()

```
PlaceBlock::PlaceBlock ( )  [default]
```

content of place block class constructor that contructs class with default values

Precondition

none

Postcondition

creates object with default values

4.13.2 Member Function Documentation

4.13.2.1 getRecord()

```
optional< Place > PlaceBlock::getRecord (
            const string & key )
```

retrieves records with matching key

Parameters

key	key to be searched

Precondition

none

Postcondition

returns object if one matches the key, returns empty object if no matching key

4.13.2.2 getSize()

```
int PlaceBlock::getSize ( )
```

returns the size of place block object

Precondition

none

Postcondition

returns int variable

4.13.2.3 print()

```
void PlaceBlock::print ( )
```

prints the records contained within

Precondition

none

Postcondition

sends contents held within to cout

4.13.2.4 sort()

```
void PlaceBlock::sort ( )
```

sorts place block content

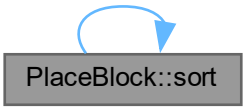
Precondition

must have content within

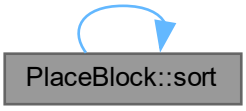
Postcondition

place block is now sorted

Here is the call graph for this function:



Here is the caller graph for this function:



4.13.3 Member Data Documentation

4.13.3.1 blockHeader

```
BlockHeader PlaceBlock::blockHeader
```

highest key

4.13.3.2 highestKey

```
string PlaceBlock::highestKey = ""  [private]
```

size of place block

4.13.3.3 placeBlock

```
vector<Place> PlaceBlock::placeBlock
```

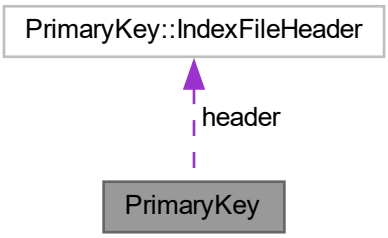
header for blocks

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

- PlaceBlock.h
- PlaceBlock.cpp

4.14 PrimaryKey Class Reference

Collaboration diagram for PrimaryKey:



Classes

- struct IndexFileHeader
- struct KeyStruct

Public Types

- enum **IndexFileFormat** { **ASCII** , **BCD** , **BINARY** }

Public Member Functions

- void GenerateIndexFile (std::string fileName)

  Generates an index file from the internal index.
- bool ReadIndexFile (std::string fileName)

  Reads an index file and stores the values in the internal index. Also records if the index's primary keys are in ascending order, to facilitate binary search.
- void Add (KeyStruct keyStruct)

  Adds a key, value pair to the index.
- int Find (std::string key)

  Performs a linear search on the internal index to try to find the given key.
- int BinarySearch (std::string str)

  Performs a binary search on the internal index to try to find the given key.

Static Public Attributes

- static const int **notFound** = -1

Private Attributes

- std::vector< KeyStruct > **vKey**
- bool **isSorted** = false
- IndexFileHeader **header**

4.14.1 Member Function Documentation

4.14.1.1 Add()

```
void PrimaryKey::Add (
            KeyStruct keyStruct )
```

Adds a key, value pair to the index.

Parameters

keyStruct	the struct containing the key, value pair to be added

4.14.1.2 BinarySearch()

```
int PrimaryKey::BinarySearch (
            std::string str )
```

Performs a binary search on the internal index to try to find the given key.

Precondition

list does not have to be sorted since the first thing the function checks is if it is, then sorts it if it is not

Postcondition

internal list is sorted in ascending order

Parameters

key	the zip code to search for

Returns

int if >= 0, the return value is the offset from the start of the file to the start of the record

int if = -1, the key could not be found in the record

4.14.1.3 Find()

```
int PrimaryKey::Find (
            std::string key )
```

Performs a linear search on the internal index to try to find the given key.

Parameters

key	the zip code to search for

Returns

int if >= 0, the return value is the offset from the start of the file to the start of the record

int if = -1, the key could not be found in the record

4.14.1.4 GenerateIndexFile()

```
void PrimaryKey::GenerateIndexFile (
            std::string fileName )
```

Generates an index file from the internal index.

Parameters

fileName	the name of the index file that will be created

4.14.1.5 ReadIndexFile()

```
bool PrimaryKey::ReadIndexFile (
            std::string fileName )
```

Reads an index file and stores the values in the internal index. Also records if the index's primary keys are in ascending order, to facilitate binary search.

Parameters

fileName	the name of the index file to be read

Returns

true file was read successfully

false file was unable to be read successfully

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

- PrimaryKey.h
- PrimaryKey.cpp

Chapter 5

File Documentation

5.1 blockBuffer.h

```
1 #include <vector>
2 #include "LengthIndicatedBuffer.h"
3 #include "Place.h"
4 #include "PlaceBlock.h"
5 using namespace std;
6
7 class BlockBuffer {
8     private:
13          bool read(istream &file);
14
15     public:
16          std::vector<char> buffer;
17          BlockFileHeader header;
18          BlockHeader blockHeader;
19          int unpackedRecords = 0;
20          int curr = 0;
21          int length = 0;
22          int RBN;
26          void processBlockHeader();
31          void setBlockHeader(BlockHeader blockHeader);
32
36          BlockBuffer() = default;
37
42          BlockBuffer(BlockFileHeader &header);
48          bool read(istream &file, int RBN);
54          void write(ostream &file, int RBN);
59          bool unpack(LengthIndicatedBuffer<BlockFileHeader> &lBuf);
65          void pack(PlaceBlock pb, LengthIndicatedBuffer<BlockFileHeader> &lBuf);
70          void pack(LengthIndicatedBuffer<BlockFileHeader> &lBuf);
75          void setPrecedingBlock(int RBN);
80          void setSucceedingBlock(int RBN);
84          void clear();
85
86          void init(BlockFileHeader &header);
87 };
```

5.2 CsvBuffer.h

```
1
4 #ifndef CSVBUFFER_H
5 #define CSVBUFFER_H
6
7 #include <istream>
8 #include <string>
9 #include <vector>
10
11 #include "enums.h"
12
13 class CsvBuffer {
14    private:
15     const char delim;
16
17     std::string buffer;
18
```

```
20      size_t curr;
21
23      // size_t recordCount;
25      size_t fieldNum;
27      size_t numFields;
28
29      // first part holds the header type for use when unpacking,
30      // second part holds the actual header value
31      std::vector<std::pair<HeaderField, std::string» headers;
32
42      void readHeader();
43
44     public:
51      CsvBuffer(const char delim = ',');
52
63      bool read(std::istream& instream);
64
79      bool unpack(std::string& str);
80
92      void init(std::istream& instream);
93
103      std::pair<HeaderField, std::string> getCurFieldHeader();
104
105      std::vector<std::pair<HeaderField, std::string» getHeaders() const;
106 };
107
108 #endif
```

5.3 enums.h

```
1
4 #ifndef ENUMS_H
5 #define ENUMS_H
6
7 // used for tiny csv parsing state machine
8 enum class CSVState {
9     QuotedField,
10      UnquotedField,
11      QuotedQuote
12 };
13
15 enum class HeaderField :  int {
16      ZipCode,
17      PlaceName,
18      State,
19      County,
20      Latitude,
21      Longitude,
22      Unknown
23 };
24
25 enum class LengthIndicatorType :  int {
26      ASCII,
27      BCD,
28      BINARY
29 };
30
31 #endif
```

5.4 Header.h

```
1
4 #ifndef HEADER_H
5 #define HEADER_H
6
7 #include <cstring>
8 #include <iostream>
9 #include <string>
10 #include <vector>
11
12 #include "enums.h"
13
14
15 // needed to remove automatic alignment of struct members
16 #pragma pack(push, 1)
17 struct HeaderInfo {
18      char magic[4];
19      int version;
20      int headerSize;
```

```
21
22      friend std::istream& operator»(std::istream& ins, HeaderInfo& headerInfo) {
23          ins.read((char*)(&headerInfo), sizeof(headerInfo));
24          return ins;
25      }
26
27      friend std::ostream& operator«(std::ostream& os, HeaderInfo& headerInfo) {
28          os.write(reinterpret_cast<char*>(&headerInfo.magic), sizeof(headerInfo.magic));
29          os.write(reinterpret_cast<char*>(&headerInfo.version), sizeof(headerInfo.version));
30          os.write(reinterpret_cast<char*>(&headerInfo.headerSize), sizeof(headerInfo.headerSize));
31          return os;
32      }
33  };
34
35  struct FileInfo {
36      int lengthIndicatorSize;
37      LengthIndicatorType lengthIndicatorFormat;
38
39      int fieldsPerRecord;
40      int primaryKeyPosition;
41
42      char indexFileName[100];
43
44      friend std::istream& operator»(std::istream& ins, FileInfo& fileInfo) {
45          ins.read((char*)(&fileInfo), sizeof(fileInfo));
46          return ins;
47      }
48
49      friend std::ostream& operator«(std::ostream& os, FileInfo& fileInfo) {
50          os.write(reinterpret_cast<char*>(&fileInfo.lengthIndicatorSize),
       sizeof(fileInfo.lengthIndicatorSize));
51          os.write(reinterpret_cast<char*>(&fileInfo.lengthIndicatorFormat),
       sizeof(fileInfo.lengthIndicatorFormat));
52
53          os.write(reinterpret_cast<char*>(&fileInfo.fieldsPerRecord), sizeof(fileInfo.fieldsPerRecord));
54          os.write(reinterpret_cast<char*>(&fileInfo.primaryKeyPosition),
       sizeof(fileInfo.primaryKeyPosition));
55
56          os.write(reinterpret_cast<char*>(&fileInfo.indexFileName), sizeof(fileInfo.indexFileName));
57
58          return os;
59      }
60  };
61
62  struct FieldInfo {
63      char fieldName[50];
64      HeaderField fieldType;
65
66      friend std::istream& operator»(std::istream& ins, FieldInfo& fieldInfo) {
67          ins.read((char*)(&fieldInfo), sizeof(fieldInfo));
68          return ins;
69      }
70
71      friend std::ostream& operator«(std::ostream& os, FieldInfo& fieldInfo) {
72          os.write(reinterpret_cast<char*>(&fieldInfo), sizeof(fieldInfo));
73          return os;
74      }
75  };
76
77  struct Header {
78      HeaderInfo headerInfo;
79      FileInfo fileInfo;
80      std::vector<FieldInfo> fields;
81
82      friend std::ostream& operator«(std::ostream& os, Header& header) {
83          os « header.headerInfo;
84          os « header.fileInfo;
85
86          for (auto f :  header.fields) {
87              os « f;
88          }
89          return os;
90      }
91  };
92
93  #pragma pack(pop)
94
95  #endif
```

5.5 HeaderBuffer.h

```
1
4  #ifndef HEADER_BUFFER_H
```

```
5 #define HEADER_BUFFER_H
6
7 #include <iostream>
8 #include <vector>
9
10 #include "Header.h"
11
12 class HeaderBuffer {
13    private:
14     std::vector<unsigned char> buffer;
15
16    public:
27     void read(std::istream& ins);
28
37     Header unpack();
38 };
39
40 #endif
```

5.6 LengthIndicatedBuffer.h

```
1
4 #ifndef LIBUFFER_H
5 #define LIBUFFER_H
6
7 #include <array>
8 #include <istream>
9 #include <string>
10 #include <vector>
11
12 #include "Header.h"
13 #include "enums.h"
14
15 class LengthIndicatedBuffer {
16    private:
17     const char delim;
18     int recordLength;
19     bool initialized = false;
20
21
22     char buffer[1000];
23
25     int curr;
26
28     size_t fieldNum = 0;
30     size_t numFields;
31
32
33    public:
34     Header header;
35
36
43     LengthIndicatedBuffer(const char delim = ',');
44
58     bool read(std::istream& instream);
59
74     bool read(std::istream& instream, int indexOffset);
75
91     bool unpack(std::string& str);
92
105     void pack(const std::string str);
106
117     void write(std::ostream& outstream);
118
132     bool init(std::istream& instream);
133
141     void writeHeader(std::ostream& outstream);
142
149     void clear();
150
162     bool checkFileType(std::istream& instream);
163
172     std::string getIndexFileName();
173
183     FieldInfo getCurFieldHeader();
184 };
185
186 #endif
```

5.7 LengthIndicatedFile.h

```
1
4 #ifndef LENGTHINDICATEDFILE_H
5 #define LENGTHINDICATEDFILE_H
6
7 #include <filesystem>
8 #include <fstream>
9 #include <iostream>
10 #include <optional>
11
12 #include "Header.h"
13 #include "LengthIndicatedBuffer.h"
14 #include "Place.h"
15 #include "PrimaryKey.h"
16 #include "enums.h"
17
18 class LengthIndicatedFile {
19    private:
20     Header header;
21     PrimaryKey index;
22
23     LengthIndicatedBuffer readBuf;
24     LengthIndicatedBuffer writeBuf;
25
26     std::string fileName;
27     std::fstream file;
28
29     int dataStart;
30
31    public:
32     LengthIndicatedFile(std::string fileName);
33     ~LengthIndicatedFile();
34     void initializeBuffers();
35     void initializeIndex();
36     bool openDataFile();
37     std::optional<Place> findRecord(std::string recordKey);
38     void generateIndex();
39     bool indexFileExists();
40 };
41
42 #endif
```

5.8 Place.h

```
1
4 #ifndef PLACE_H
5 #define PLACE_H
6
7 #include <string>
8
9 #include "CsvBuffer.h"
10 #include "LengthIndicatedBuffer.h"
11
12 /* This file contains the record details like zipcode, State ID, longitude and latitude.
13 We have the unpack and read functions that
14 puts the data from CSV file into the buffer and reads the data from CSV file.  */
15 class Place {
16    public:
17     Place();
18
24     Place(const Place& loc);
25
31     std::string getZipCode() const;
32
38     std::string getState() const;
39
45     std::string getName() const;
46
52     std::string getCounty() const;
53
59     double getLat() const;
60
66     double getLongi() const;
67
77     void unpack(CsvBuffer& buffer);
78
88     void unpack(LengthIndicatedBuffer& buffer);
89
90     void pack(LengthIndicatedBuffer& buffer);
91
97     void operator=(const Place& loc);
98
```

```
104     size_t getSize();
105
106     void print();
107
108    private:
109     std::string zipcode;
110     std::string name;
111     std::string state;
112     std::string county;
113     double latitude;
114     double longitude;
115 };
116
117 #endif
```

5.9 PlaceBlock.h

```
1 #include <optional>
2 #include "BlockBuffer.h"
3 #include "Place.h"
4 using namespace std;
5
6 class PlaceBlock {
7    private:
8     int size = -1;
9     string highestKey = "";
11    public:
12     BlockHeader blockHeader;
13     vector<Place> placeBlock;
17     PlaceBlock() = default;
18
19     bool unpack(BlockBuffer& bBuf, LengthIndicatedBuffer<BlockFileHeader>& lBuf);
20     string getHighestKey();
25     optional<Place> getRecord(const string& key);
29     void print();
33     void sort();
37     int getSize();
38 };
```

5.10 PrimaryKey.h

```
1
4 #pragma once
5 #include <fstream>
6
7 #include "Header.h"
8
9 class PrimaryKey {
10
11    public:
12     static const int notFound = -1;
13     enum IndexFileFormat {
14         ASCII,
15         BCD,
16         BINARY
17     };
18      struct IndexFileHeader {
19         int version;
20         int keyCount;
21         int format;
22     };
23     struct KeyStruct {
24         std::string key;
25         unsigned int offset;
26     };
32     void GenerateIndexFile(std::string fileName);
33
42     bool ReadIndexFile(std::string fileName);
43
49     void Add(KeyStruct keyStruct);
50
58     int Find(std::string key);
59
70     int BinarySearch(std::string str);
71
72    private:
73     std::vector<KeyStruct> vKey;
74     bool isSorted = false;
75     IndexFileHeader header;
76 };
```