Sequence Set

1.0

Generated by Doxygen 1.8.16

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:

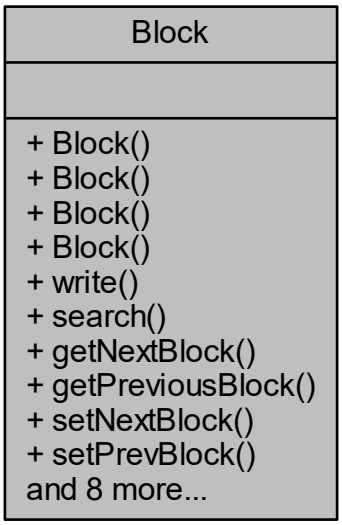
Chapter 3

Class Documentation

3.1 Block Class Reference

`#include <Block.h>`

Collaboration diagram for Block:



Public Member Functions

- Block ()

  Default constructor.
- Block (unsigned long long _RBN)

  Relative Block Number constructor.

- Block (string[ ])
  Constructor with record numbers.
- Block (string)
  Constructor with record numbers.
- void write (string)
- int search (string pKey)
  Searches for record.
- Block * getNextBlock ()
  Gets pointer of next block.
- Block * getPreviousBlock ()
  Gets pointer of previous block.
- void setNextBlock (Block *nextBlockPtr)
  Sets pointer to next block.
- void setPrevBlock (Block *previousBlockPtr)
  Sets pointer to previous block.
- int getRecordCount ()
  Gets the record count.
- int getLastRecordPKey ()
  Gets the last record of the block.
- bool deleteRecord (string pKey)
- bool addRecord (string pKey)
- void getRecords (Record block[ ])
- string blockData ()
  Returns RBN and records of the block.
- unsigned long long getRBN ()
  Gets the relative block number.
- void setRBN (unsigned long long)
  Set rleative block number.

3.1.1 Detailed Description

Definition at line 30 of file Block.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Block() [1/4]

```
Block::Block ( )
```

Default constructor.

Precondition

None

Postcondition

A blank Block object is created

Definition at line 40 of file Block.cpp.

3.1.2.2 Block() [2/4]

```
Block::Block (
            unsigned long long _RBN )
```

Relative Block Number constructor.

Precondition

None

Postcondition

A blank Block object is created

Definition at line 55 of file Block.cpp.

3.1.2.3 Block() [3/4]

```
Block::Block (
            string [] )
```

Constructor with record numbers.

Precondition

The passed array must be of size fill count

Postcondition

A block object is made using an array of primary keys

3.1.2.4 Block() [4/4]

```
Block::Block (
            string _blockData )
```

Constructor with record numbers.

Precondition

A string

Postcondition

A Block object is created using the string

Definition at line 80 of file Block.cpp.

3.1.3 Member Function Documentation

3.1.3.1 write()

```
void Block::write (
            string _fileName )
```

Precondition

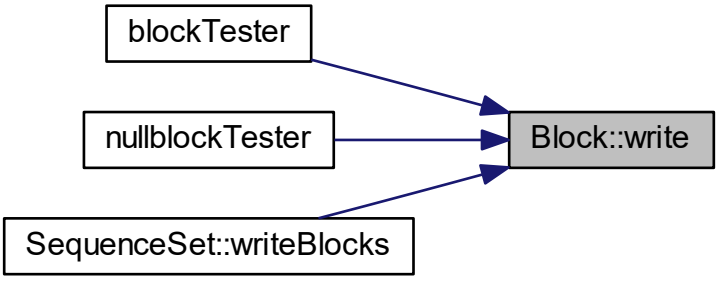
A block

Postcondition

Writes the block to a file

Definition at line 146 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.2 search()

```
int Block::search (
            string pKey )
```

Searches for record.

Precondition

Primary key

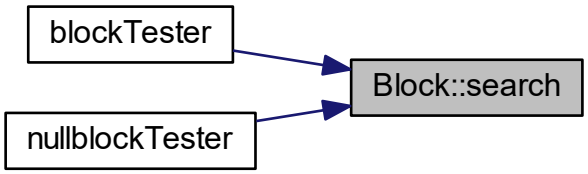
Postcondition

Returns the record or 0 if the record is not found

Definition at line 185 of file Block.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



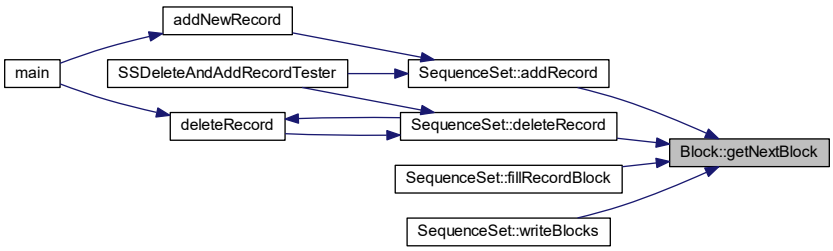
3.1.3.3 getNextBlock()

```
Block * Block::getNextBlock ( )
```

Gets pointer of next block.

Definition at line 192 of file Block.cpp.

Here is the caller graph for this function:



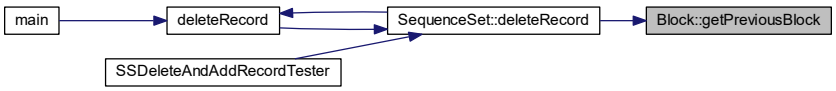
3.1.3.4 getPreviousBlock()

```
Block * Block::getPreviousBlock ( )
```

Gets pointer of previous block.

Definition at line 199 of file Block.cpp.

Here is the caller graph for this function:



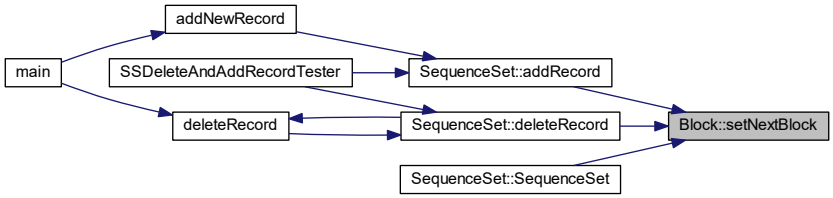
3.1.3.5 setNextBlock()

```
void Block::setNextBlock (
            Block * nextBlockPtr )
```

Sets pointer to next block.

Definition at line 206 of file Block.cpp.

Here is the caller graph for this function:



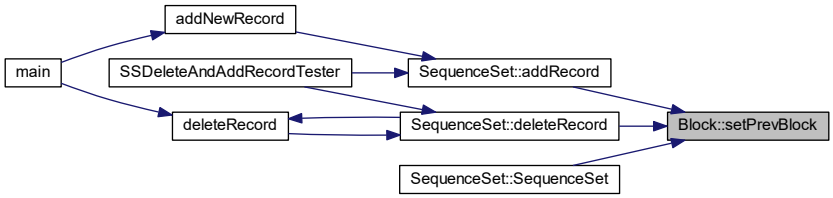
3.1.3.6 setPrevBlock()

```
void Block::setPrevBlock (
            Block * previousBlockPtr )
```

Sets pointer to previous block.

Definition at line 213 of file Block.cpp.

Here is the caller graph for this function:



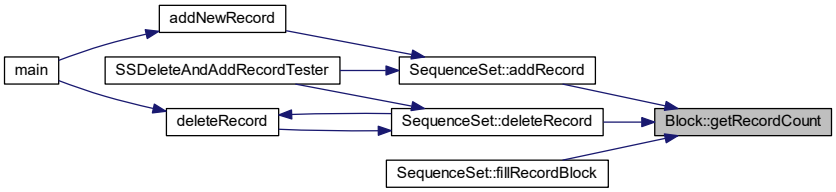
3.1.3.7 getRecordCount()

```
int Block::getRecordCount ( )
```

Gets the record count.

Definition at line 220 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.8 getLastRecordPKey()

```
int Block::getLastRecordPKey ( )
```

Gets the last record of the block.

Definition at line 225 of file Block.cpp.

Here is the caller graph for this function:

3.1.3.9 deleteRecord()

```
bool Block::deleteRecord (
            string pKey )
```

Precondition

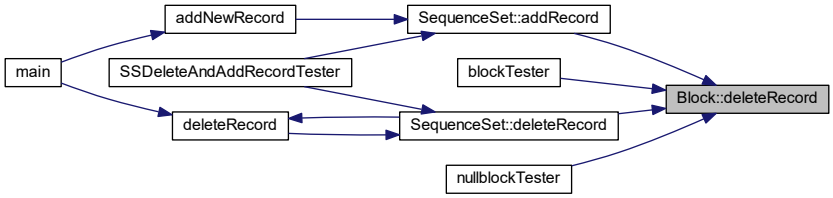
Primary key

Postcondition

Deletes the record with the given primary key

Definition at line 231 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.10 addRecord()

```
bool Block::addRecord (
            string pKey )
```

Precondition

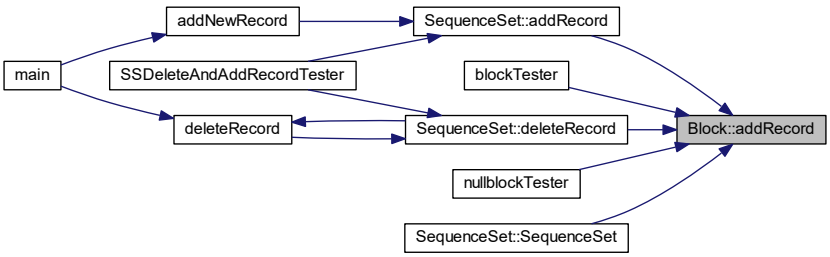
Primary key

Postcondition

Adds the record with the given primary key

Definition at line 260 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.11 getRecords()

```
void Block::getRecords (
            Record block[] )
```

Precondition

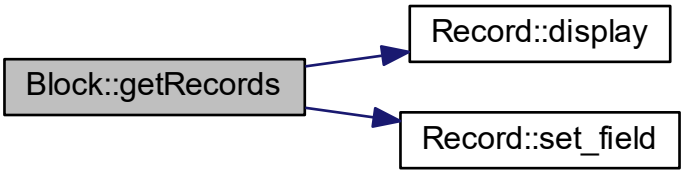
Record object array

Postcondition

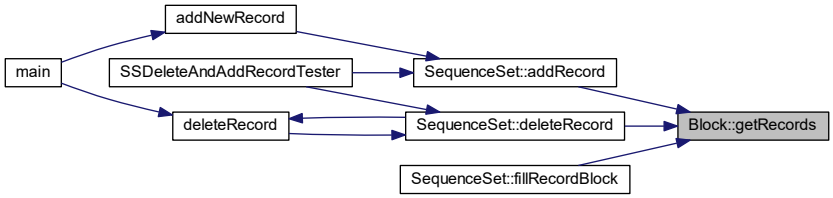
Fills record block

Definition at line 289 of file Block.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.12 blockData()

```
string Block::blockData ( )
```

Returns RBN and records of the block.

Definition at line 70 of file Block.cpp.

3.1.3.13 getRBN()

```
unsigned long long Block::getRBN ( )
```

Gets the relative block number.

Definition at line 302 of file Block.cpp.

3.1.3.14 setRBN()

```
void Block::setRBN (
            unsigned long long RBN )
```

Set rleative block number.

Precondition

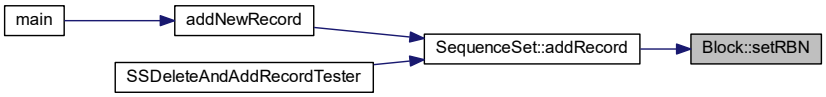
unsigned long long int

Postcondition

Sets the relative block number

Definition at line 306 of file Block.cpp.

Here is the caller graph for this function:



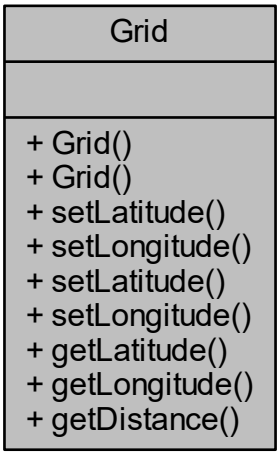
The documentation for this class was generated from the following files:

- C:/CSCI/331/Doxygen/Input/Block.h
- C:/CSCI/331/Doxygen/Input/Block.cpp

3.2 Grid Class Reference

Grid class.

Collaboration diagram for Grid:



Public Member Functions

- Grid ()

  Default constructor.
- Grid (float, float)

  Constructor requiring both latitude and longitude.
- void setLatitude (float)

  Sets Latitude for this grid object.
- void setLongitude (float)

  Sets Longitude for this grid object.
- void setLatitude (string)

  Sets Latitude for this grid object.
- void setLongitude (string)

  Sets Longitude for this grid object.
- float getLatitude ()

  Gets Latitude for this grid object.
- float getLongitude ()

  Gets Longitude for this grid object.
- float getDistance (Grid)

  Gets Distance from this grid object to another grid object.

3.2.1 Detailed Description

Grid class.

Variables for latitude and longitude, constructor for setting 0 to both latitude and longitude (default constructor) and a constructor for setting latitude and longitude to input values.

Methods for setting and getting latitude and longitude and for getting the distance between two points.

Definition at line 30 of file grid.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Grid() [1/2]

```
Grid::Grid ( )
```

Default constructor.

Precondition

none

Postcondition

sets values for latitude and longitude to 0

Definition at line 51 of file grid.cpp.

3.2.2.2 Grid() [2/2]

```
Grid::Grid (
            float _latitude,
            float _longitude )
```

Constructor requiring both latitude and longitude.

Precondition

Values for latitude and longitude as float

Postcondition

Sets values for latitude and longitude

Definition at line 60 of file grid.cpp.

3.2.3 Member Function Documentation

3.2.3.1 setLatitude() [1/2]

```
void Grid::setLatitude (
            float _latitude )
```

Sets Latitude for this grid object.

Precondition

_latitude must follow rules regarding floats

Postcondition

Sets latitude for grid object

Definition at line 69 of file grid.cpp.

3.2.3.2 setLongitude() [1/2]

```
void Grid::setLongitude (
            float _longitude )
```

Sets Longitude for this grid object.

Precondition

_longitude must follow rules regarding floats

Postcondition

Sets longitude for grid object

Definition at line 85 of file grid.cpp.

3.2.3.3 setLatitude() [2/2]

```
void Grid::setLatitude (
            string _latitude )
```

Sets Latitude for this grid object.

Precondition

_latitude must follow rules regarding string to float

Postcondition

Sets latitude for grid object

Definition at line 77 of file grid.cpp.

3.2.3.4 setLongitude() [2/2]

```
void Grid::setLongitude (
            string _longitude )
```

Sets Longitude for this grid object.

Precondition

_longitude must follow rules regarding string to float

Postcondition

Sets longitude for grid object

Definition at line 93 of file grid.cpp.

3.2.3.5 getLatitude()

```
float Grid::getLatitude ( )
```

Gets Latitude for this grid object.

Precondition

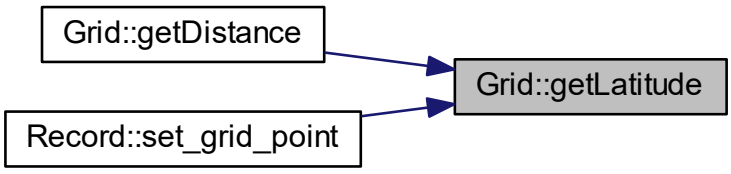
none

Postcondition

returns latitude for grid object as float

Definition at line 101 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.6 getLongitude()

```
float Grid::getLongitude ( )
```

Gets Longitude for this grid object.

Precondition

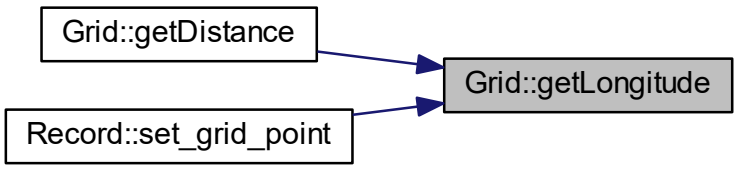
none

Postcondition

returns longitude for grid object as float

Definition at line 109 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.7 getDistance()

```
float Grid::getDistance (
            Grid _grid )
```

Gets Distance from this grid object to another grid object.

Precondition

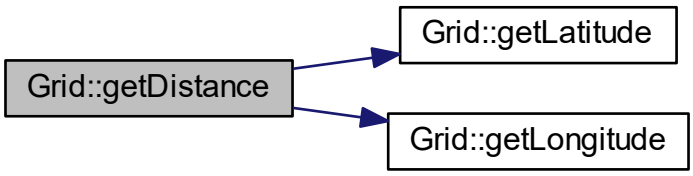
grid object must be provided

Postcondition

returns distance from this grid object to another grid object as float

Definition at line 117 of file grid.cpp.

Here is the call graph for this function:



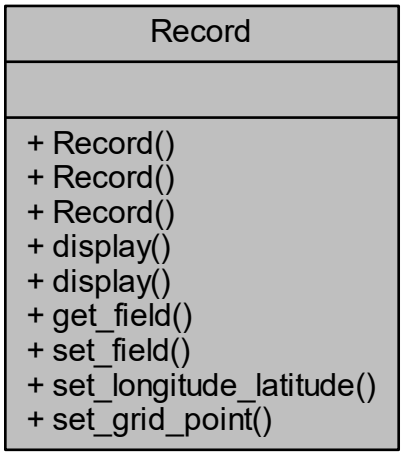
The documentation for this class was generated from the following file:

- C:/CSCI/331/Doxygen/Input/grid.cpp

3.3 Record Class Reference

`#include <Record.h>`

Collaboration diagram for Record:



Public Member Functions

- Record ()

  Default constructor.
- Record (string, string, string, string, Grid)

  Constructor with a grid object.
- Record (string, string, string, string, string, string)

  Constructor that also takes latitude, and longitude.
- void display ()

  Displays all fields of the record.
- void display (string)

  Displays the specified field.
- string get_field (string)

  Get the desired field in the record to display a field from its data.
- void set_field (string, string)
- void set_longitude_latitude (float, float)

  Sets the latitude and longitude.
- void set_grid_point (Grid)

  Sets the Latitude and longitude based on a grid point.

3.3.1 Detailed Description

Definition at line 25 of file Record.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Record() [1/3]

```
Record::Record ( )
```

Default constructor.

Precondition

None

Postcondition

A blank record object is created

Definition at line 22 of file Record.cpp.

3.3.2.2 Record() [2/3]

```
Record::Record (
            string _zip_code,
            string _place_name,
            string _state,
            string _county,
            Grid _gridPoint )
```

Constructor with a grid object.

Precondition

Grid object is provided

Postcondition

A filled record object is created with a grid object

Definition at line 32 of file Record.cpp.

3.3.2.3 Record() [3/3]

```
Record::Record (
            string _zip_code,
            string _place_name,
            string _state,
            string _county,
            string latitude,
            string longitude )
```

Constructor that also takes latitude, and longitude.

Precondition

String is provided in order of latitude, longitude

Postcondition

A filled record object is created with a latitude and longitude

Definition at line 42 of file Record.cpp.

3.3.3 Member Function Documentation

3.3.3.1 display() [1/2]

```
void Record::display ( )
```

Displays all fields of the record.

Precondition

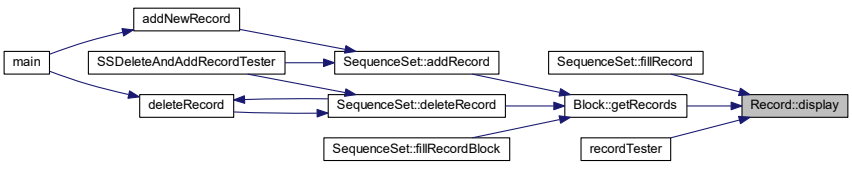
None

Postcondition

Record object will display all of its own data

Definition at line 71 of file Record.cpp.

Here is the caller graph for this function:



3.3.3.2 display() [2/2]

```
void Record::display (
            string field )
```

Displays the specified field.

Precondition

None

Postcondition

Record object will display specified field

Definition at line 84 of file Record.cpp.

3.3.3.3 get_field()

```
string Record::get_field (
            string field )
```

Get the desired field in the record to display a field from its data.

Precondition

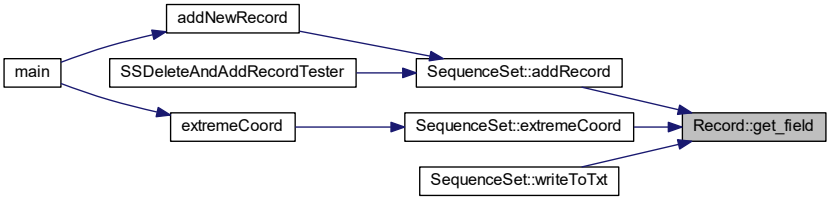
Provided string must match the name of a field in the record

Postcondition

Record object will display the specified field from its own data

Definition at line 109 of file Record.cpp.

Here is the caller graph for this function:



3.3.3.4 set_field()

```
void Record::set_field (
            string field,
            string data )
```

Precondition

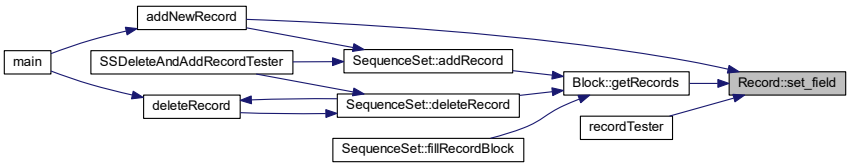
First provided string must match the name of a field in the record Second provided string must be the appropriate length for the field

Postcondition

Record object will display the specified field from its own data

Definition at line 137 of file Record.cpp.

Here is the caller graph for this function:



3.3.3.5 set_longitude_latitude()

```
void Record::set_longitude_latitude (
            float longitude,
            float latitude )
```

Sets the latitude and longitude.

Precondition

Provide longitude and latitude as floats

Postcondition

Set the latitude and longitude of the record

Definition at line 166 of file Record.cpp.

3.3.3.6 set_grid_point()

```
void Record::set_grid_point (
            Grid _gridPoint )
```

Sets the Latitude and longitude based on a grid point.

Precondition

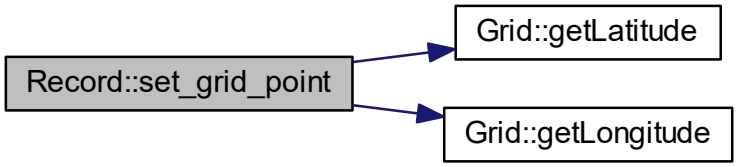
A grid point of type Grid

Postcondition

Sets latitude and longitude based on grid point recieved

Definition at line 172 of file Record.cpp.

Here is the call graph for this function:



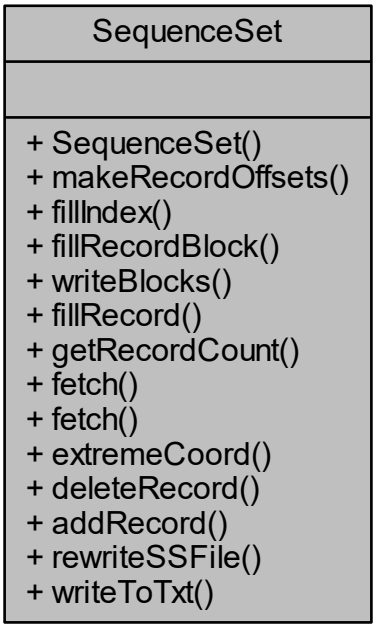
The documentation for this class was generated from the following files:

- C:/CSCI/331/Doxygen/Input/Record.h
- C:/CSCI/331/Doxygen/Input/Record.cpp

3.4 SequenceSet Class Reference

```
#include <SequenceSet.h>
```

Collaboration diagram for SequenceSet:



Public Member Functions

- DATAFILENAME and HEADERENDSTRING SequenceSet ()

  Default constructor.
- void makeRecordOffsets (string fileName)

  Make record offsets.
- void fillIndex ()

  Fill Index.
- void fillRecordBlock (unsigned long long blockID)

  Fill record block.
- void writeBlocks ()

  Write blocks.
- Record fillRecord (string RecordString)

  Fill record.
- unsigned int getRecordCount ()

  Get record count.
- string fetch (string pKey)

  Fetch string.
- string fetch (unsigned int pKey)

  Fetch unsigned int.
- string extremeCoord (string, char)

  Extreme coordinate.
- bool deleteRecord (int pKey)

Delete record.

- void addRecord (Record record)

  Add record.
- void rewriteSSFile ()
- void writeToTxt (Record, string, string)

3.4.1 Detailed Description

Definition at line 32 of file SequenceSet.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 SequenceSet()

```
SequenceSet::SequenceSet ( )
```

Default constructor.

Precondition

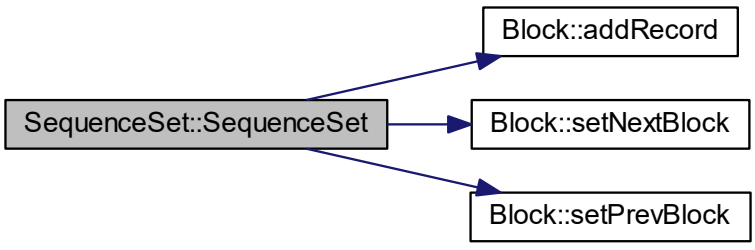
File named DATAFILENAME

Postcondition

Returns true if found otherwise returns false

Definition at line 30 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 makeRecordOffsets()

```
void SequenceSet::makeRecordOffsets (
            string fileName )
```

Make record offsets.

Precondition

File must have fixed length primary keys equal to the "ziplength" in globals.cpp

Postcondition

An index file is made for the provided file name

Definition at line 180 of file SequenceSet.cpp.

3.4.3.2 fillIndex()

```
void SequenceSet::fillIndex ( )
```

Fill Index.

Precondition

"RecordOffsets.txt" file must exist makeRecordOffsets can be ran to be sure of this

Postcondition

The index is made and stored here, in the Sequence Set

Definition at line 119 of file SequenceSet.cpp.

3.4.3.3 fillRecordBlock()

```
void SequenceSet::fillRecordBlock (
            unsigned long long blockID )
```

Fill record block.

Precondition

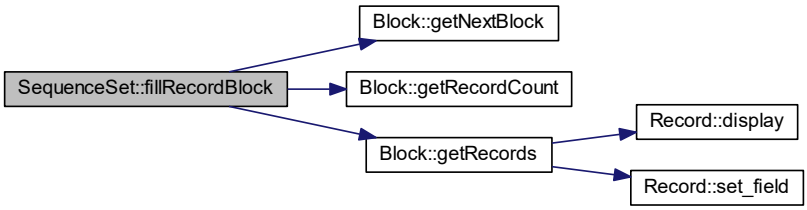
blockID must be less than the block count

Postcondition

Block is loaded into a record block

Definition at line 332 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.4 writeBlocks()

```
void SequenceSet::writeBlocks ( )
```

Write blocks.

Precondition

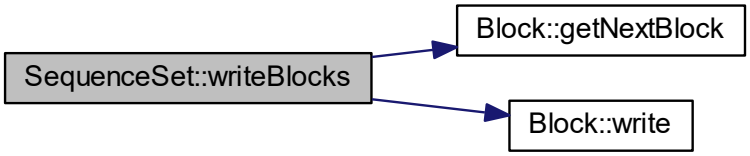
None

Postcondition

All blocks are called to run their write function

Definition at line 323 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.5 fillRecord()

```
Record SequenceSet::fillRecord (
            string RecordString )
```

Fill record.

Precondition

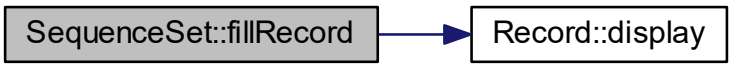
Record string must follow parameter conventions Record string must be complete, call fetch if needed

Postcondition

A record string is loaded into a record object

Definition at line 259 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.6 getRecordCount()

```
unsigned int SequenceSet::getRecordCount ( )
```

Get record count.

Precondition

Files must be available and the header in data file must contain "Records:"

Postcondition

RecordCount is returned

Definition at line 85 of file SequenceSet.cpp.

3.4.3.7 fetch() [1/2]

```
string SequenceSet::fetch (
            string pKey )
```

Fetch string.

Precondition

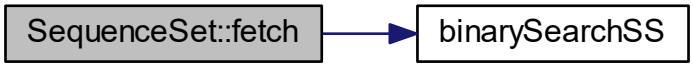
None

Postcondition

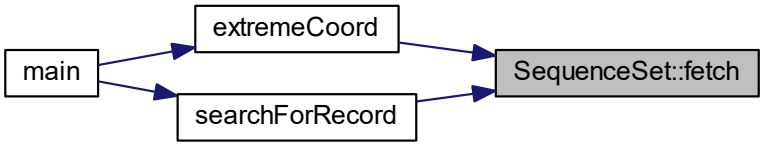
returns the whole record as a string

Definition at line 151 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.8 fetch() [2/2]

```
string SequenceSet::fetch (
            unsigned int pKey )
```

Fetch unsigned int.

Precondition

None

Postcondition

returns the whole record as a string

Definition at line 176 of file SequenceSet.cpp.

3.4.3.9 extremeCoord()

```
string SequenceSet::extremeCoord (
            string state,
            char direction )
```

Extreme coordinate.

Precondition

State of type string and Direction of type Char (N, E, S, W) State code must be in the list of states or the last state in list is used

Postcondition

Returns the zipcode containing the most extreme point of said direction

Definition at line 517 of file SequenceSet.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

3.4.3.10 deleteRecord()

```
bool SequenceSet::deleteRecord (
            int pKey )
```

Delete record.

Precondition

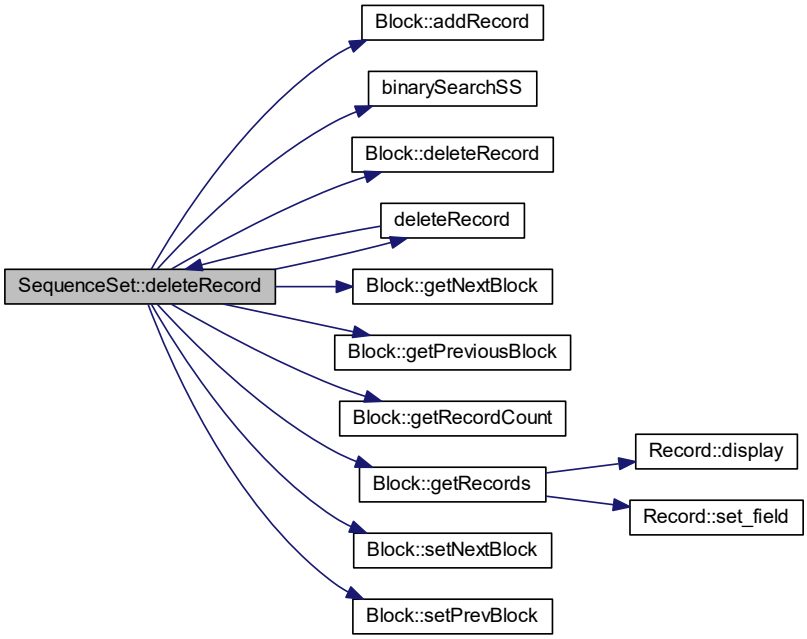
A primary key (zipcode)

Postcondition

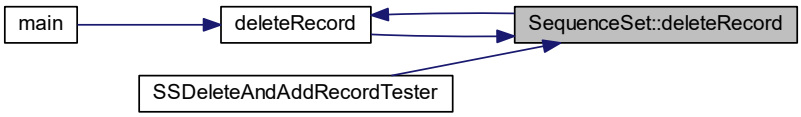
Deletes record with given zipcode

Definition at line 412 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.11 addRecord()

```
void SequenceSet::addRecord (
            Record record )
```

Add record.

Precondition

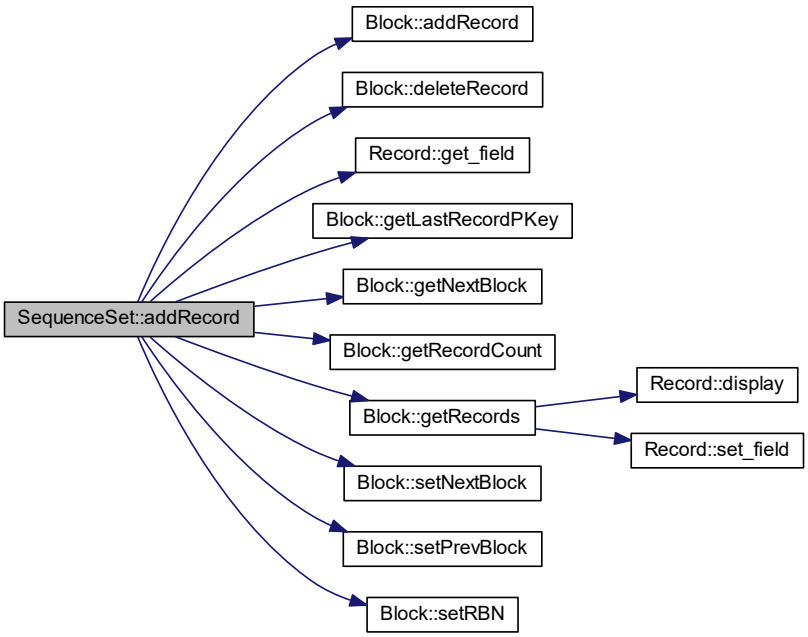
A record object

Postcondition

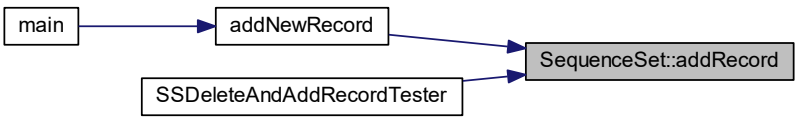
Adds the record

Definition at line 673 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

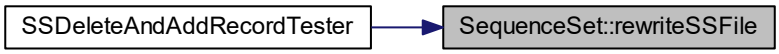


3.4.3.12 rewriteSSFile()

```
void SequenceSet::rewriteSSFile ( )
```

Definition at line 795 of file SequenceSet.cpp.

Here is the caller graph for this function:

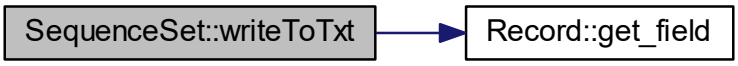


3.4.3.13 writeToTxt()

```
void SequenceSet::writeToTxt (
            Record record,
            string offset,
            string _fileName )
```

Definition at line 807 of file SequenceSet.cpp.

Here is the call graph for this function:



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

- C:/CSCI/331/Doxygen/Input/SequenceSet.h
- C:/CSCI/331/Doxygen/Input/SequenceSet.cpp

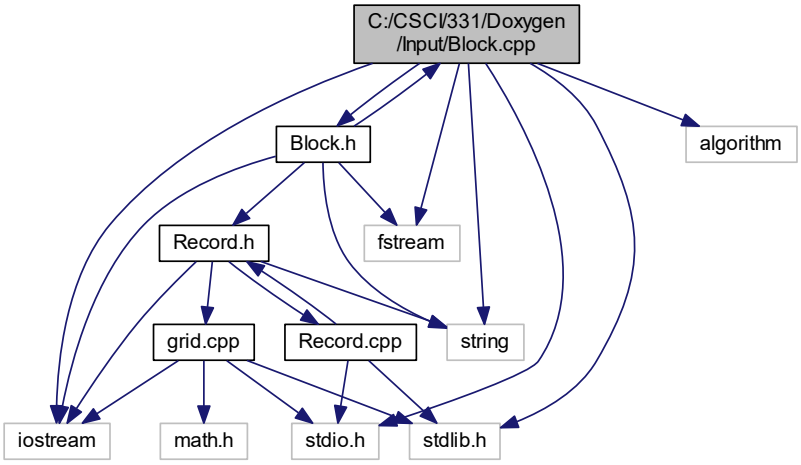
Chapter 4

File Documentation

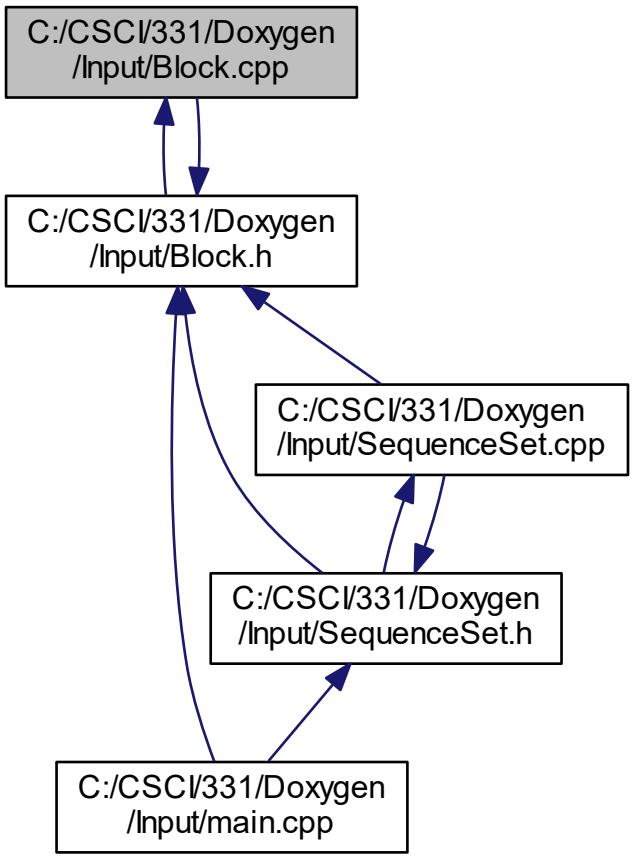
4.1 C:/CSCI/331/Doxygen/Input/Block.cpp File Reference

```
#include "Block.h"
#include <iostream>
#include <fstream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
```

Include dependency graph for Block.cpp:



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



Functions

- int binarySearch (const string arr[ ], string x, int n)

  Searches block for record by primary key.
- void convertStrArrToIntArr (const string strArr[ ], int intArr[ ], int ArrLength)

  String to integer.
- void convertIntArrToStrArr (string strArr[ ], int intArr[ ], int ArrLength)

  Integer to string.

Variables

- const string null_str = ""
- const int NULL_INT = 1000000

4.1.1 Function Documentation

4.1.1.1 binarySearch()

```
int binarySearch (
            const string arr[],
            string x,
            int n )
```

Searches block for record by primary key.

Precondition

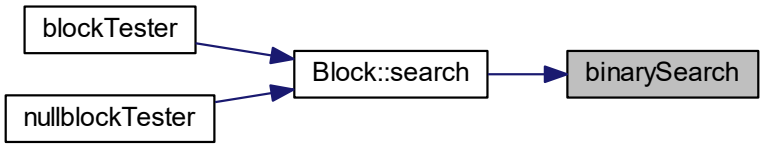
Primary key

Postcondition

Returns true if found otherwise returns false

Definition at line 328 of file Block.cpp.

Here is the caller graph for this function:



4.1.1.2 convertStrArrToIntArr()

```
void convertStrArrToIntArr (
            const string strArr[],
            int intArr[],
            int ArrLength )
```

String to integer.

Precondition

An array of strings

Postcondition

An array of integers

Definition at line 377 of file Block.cpp.

4.1.1.3 convertIntArrToStrArr()

```
void convertIntArrToStrArr (
             string strArr[],
             int intArr[],
             int ArrLength )
```

Integer to string.

Precondition

An array of integers

Postcondition

An array of strings

Definition at line 393 of file Block.cpp.

4.1.2 Variable Documentation

4.1.2.1 null_str

```
const string null_str = ""
```

Definition at line 37 of file Block.cpp.

4.1.2.2 NULL_INT

```
const int NULL_INT = 1000000
```

Definition at line 38 of file Block.cpp.

4.2 Block.cpp

```
00001
00022 #include "Block.h"
00023 #include <iostream>
00024 #include <fstream>
00025 #include <string>
00026 #include <stdio.h>
00027 #include <stdlib.h>
00028 #include <algorithm>
00029
00030 using namespace std;
00031
00032 //prototype for binary search
00033 int binarySearch(const string[], string,int);
00034 void convertStrArrToIntArr(const string[], int[], int);
00035 void convertIntArrToStrArr(string [], int [], int );
00036
00037 const string null_str = "";
00038 const int NULL_INT = 1000000;
00039
00040 Block::Block()
00041 {
00042   isEmpty = true;
00043   relativeBlockNumber = 0;
00044   recordCount = 0;
00045   for(int i = 0; i < RECORDSPERBLOCK; i++){
00046     records[i] = "";
00047   }
00048
00049   nextBlock = nullptr;
00050   previousBlock = nullptr;
00051
00052   if(DEBUG) {cout « "Made an empty block.\n";}
00053 }
00054
00055 Block::Block(unsigned long long _RBN)
00056 {
00057   isEmpty = true;
00058   relativeBlockNumber = _RBN;
00059   recordCount = 0;
00060   for(int i = 0; i < RECORDSPERBLOCK; i++){
00061     records[i] = "";
00062   }
00063
00064   nextBlock = nullptr;
00065   previousBlock = nullptr;
00066
00067   if(DEBUG) {cout « "Made an empty block.\n";}
00068 }
00069
00070 string Block::blockData(){
00071   string returnString = "";
00072   returnString += relativeBlockNumber;
00073   for(int i = 0; i < recordCount; i++){
00074     returnString += " ";
00075     returnString += records[i];
00076   }
00077   return returnString;
00078 }
00079
00080 Block::Block(string _blockData)
00081 {
00082   if(DEBUG) {cout « "Making a block with \"" « _blockData « "\".\n";}
00083
00084   isEmpty = false;
00085   relativeBlockNumber = 0;
00086   recordCount = 0;
00087
00088   //set the primary keys of each record
00089   string tempStr = "";
00090   int recordNumber = 0;
00091   int j = 0; //pointer to track the position in the string
00092   while( j < _blockData.length() && j < BLOCKFILLCOUNT*ZIPLENGTH)
00093   {
00094     for(int i = 0; i < ZIPLENGTH; i++) //for each element of the pKey
00095     {
00096       if( _blockData[j] >= '0' && _blockData[j] <= '9' )
00097       {
00098         tempStr += _blockData[j]; //if the element is numeric, store the value
00099       }
00100       j++; //increment the pointer
00101     }
00102     records[recordNumber] = tempStr; //store the pKey in the class
00103     tempStr = ""; //clear the temp string
00104     if(records[recordNumber] != ""){
00105       recordCount++; //update the number of records in the block
```

```
00106         recordNumber++; //increment the record number
00107     }
00108   }
00109
00110   if(DEBUG) {cout « "Elements of Constructed block " « relativeBlockNumber « ": \"" ;
00111              for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i] « " ";}
00112              cout « "\".\n";}
00113
00114   nextBlock = nullptr;
00115   previousBlock = nullptr;
00116 }
00117
00118 Block::Block(string _blockData[RECORDSPERBLOCK])
00119 {
00120   if(DEBUG) {cout « "Making a block with \"" ;
00121              for(int i = 0; i < BLOCKFILLCOUNT; i++){cout « _blockData[i];}
00122              cout « "\".\n";}
00123
00124   isEmpty = false;
00125   relativeBlockNumber = 0;
00126   recordCount = 0;
00127
00128   //set the primary keys of each record
00129   for(int i = 0; i < BLOCKFILLCOUNT; i++)
00130   {
00131     records[i] = _blockData[i];
00132     if(records[i] != ""){
00133       recordCount++;
00134     }
00135   }
00136
00137   if(DEBUG) {cout « "Elements of Constructed block " « relativeBlockNumber « ": \"" ;
00138              for(int i = 0; i < BLOCKFILLCOUNT; i++){cout « records[i] « " ";}
00139              cout « "\".\n";}
00140
00141   nextBlock = nullptr;
00142   previousBlock = nullptr;
00143 }
00144
00145
00146 void Block::write(string _fileName)
00147 {
00148   ofstream file;
00149   file.open(_fileName, ios_base::app);  if(DEBUG) {cout « "Writing block number " «
       relativeBlockNumber « " to a "« _fileName «".\n";}
00150
00151   file.seekp(relativeBlockNumber * BLOCKLENGTH);
00152
00153   if(DEBUG){
00154     cout « relativeBlockNumber « ": ";
00155     for(int i = 0; i < recordCount; i++){
00156       cout « records[i] « " ";
00157     }
00158     cout « endl;
00159   }
00160
00161     file « "RBN " « relativeBlockNumber « ": ";
00162   if(DEBUG)(cout « "The file should read: \"");
00163     if(DEBUG){cout « "RBN " « relativeBlockNumber « ": ";}
00164     for(int i = 0;  i < recordCount; i++){
00165         string record = records[i];
00166         for(int j = ZIPLENGTH - record.length(); j > 0; j--){
00167             file « " ";
00168             if(DEBUG){cout « " ";}
00169         }
00170
00171         file « record;
00172         if(DEBUG){cout « record;}
00173     }
00174   for(int i = RECORDSPERBLOCK - recordCount; i > 0; i--){
00175     for(int j = 0; j < ZIPLENGTH; j++){
00176       file « " ";
00177       if(DEBUG){cout « " ";}
00178     }
00179   }
00180   file « "\n";
00181   if(DEBUG)(cout « "\"\n");
00182   file.close();
00183 }
00184
00185 int Block::search(string pKey)
00186 {
00187     if(DEBUG) {cout « "Searching for " « pKey « " in this block\n";}
00188
00189   return binarySearch(records, pKey, RECORDSPERBLOCK );
00190 }
00191
```

```
00192 Block * Block::getNextBlock()
00193 {
00194   return nextBlock;
00195
00196     if(DEBUG) {cout « "Pointer to the next block has been returned.\n";}
00197 }
00198
00199 Block * Block::getPreviousBlock()
00200 {
00201   return previousBlock;
00202
00203     if(DEBUG) {cout « "Pointer to the previous block has been returned.\n";}
00204 }
00205
00206 void Block::setNextBlock( Block * nextBlockPtr )
00207 {
00208   nextBlock = nextBlockPtr;
00209
00210   if(DEBUG) {cout « "Pointer to the next block has been set.\n";}
00211 }
00212
00213 void Block::setPrevBlock( Block * previousBlockPtr )
00214 {
00215   previousBlock = previousBlockPtr;
00216
00217   if(DEBUG) {cout « "Pointer to the previous block has been set.\n";}
00218 }
00219
00220 int Block::getRecordCount()
00221 {
00222   return recordCount;
00223 }
00224
00225 int Block::getLastRecordPKey()
00226 {
00227   if(DEBUG) {cout « "Getting last record of the block\n";}
00228   return stoi( records[ recordCount - 1 ] );
00229 }
00230
00231 bool Block::deleteRecord(string pKey)
00232 {
00233   if(DEBUG) {cout « "Deleting record " « pKey « " in block "« relativeBlockNumber «"\n";}
00234   if(DEBUG) {cout « "Elements of Constructed block before deleting record: \"" ;
00235       for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
00236       cout « "\".\n";}
00237
00238   int position = this -> search(pKey); //get the position of the record to be deleted
00239
00240   if ( position != -1 )
00241   {
00242     records[position] = ""; //delete the record
00243     recordCount--;  //decrement record count
00244     if(DEBUG) {cout « "Elements of Constructed block after deleting record: \"" ;
00245         for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout « "null";}else{cout «
     records[i];}}
00246         cout « "\".\n";}
00247     sortRecord(); //sort the record
00248     if(DEBUG) {cout « "Elements of Constructed block after sorting record: \"" ;
00249         for(int i = 0; i < RECORDSPERBLOCK; i++){if(records[i] == null_str){cout « "null";}else{cout «
     records[i];}}
00250         cout « "\".\n";}
00251     return true;
00252   }
00253     else
00254     {
00255       if(DEBUG) {cout « "Record not found in block. Could not delete"  « "\".\n";}
00256       return false;
00257     }
00258 }
00259
00260 bool Block::addRecord(string pKey)
00261 {
00262   if(DEBUG) {cout « "Adding a record to "« relativeBlockNumber «".\n";}
00263   if(DEBUG) {cout « "Elements of Constructed block before adding record: \"" ;
00264       for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
00265       cout « "\".\n";}
00266
00267   for(int i = 0; i < RECORDSPERBLOCK; i++) //go through the block to see if there is empty record
00268   {
00269     if( records[i] == null_str) //if there is an empty record
00270     {
00271       records[i] = pKey;  //fill the record with the pKey
00272       recordCount++;  //increment record count
00273       if(DEBUG) {cout « "Elements of Constructed block after adding record: \"" ;
00274             for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
00275             cout « "\".\n";}
00276       sortRecord(); //sort the record
```

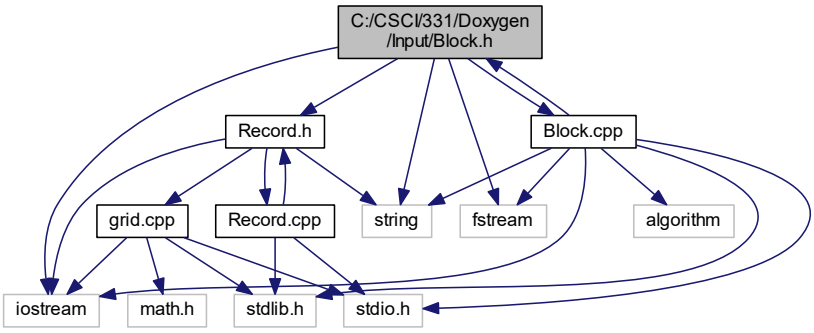
```
00277       if(DEBUG) {cout « "Elements of Constructed block after sorting record: \"" ;
00278             for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
00279             cout « "\".\n";}
00280       return true;
00281     }
00282   }
00283
00284   if(DEBUG) {cout « "Block Full. Could not add record."  « "\".\n";}
00285
00286   return false;
00287 }
00288
00289 void Block::getRecords(Record block[])
00290 {
00291   if(DEBUG) {cout « "Setting record zips\n";}
00292
00293   for(auto i = 0; i < RECORDSPERBLOCK; i++){
00294     block[i].set_field("ZIP", records[i]);
00295     if(DEBUG){
00296       cout« "Block["«i«"]: " « endl;
00297       block[i].display();
00298     }
00299   }
00300 }
00301
00302 unsigned long long Block::getRBN(){
00303   return relativeBlockNumber;
00304 }
00305
00306 void Block::setRBN(unsigned long long RBN){
00307   relativeBlockNumber = RBN;
00308 }
00309
00310 void Block::sortRecord()
00311 {
00312   if(DEBUG) {cout « "Sorting the records in the block.\n";}
00313
00314   int int_records_array[RECORDSPERBLOCK]; //to convert the string of records to integers
00315   convertStrArrToIntArr(records, int_records_array, RECORDSPERBLOCK);
00316
00317   int n = sizeof(int_records_array)/sizeof(int_records_array[0]);
00318   sort(int_records_array, int_records_array+n);
00319
00320   //convert back to strings and store in records array of string
00321   convertIntArrToStrArr(records, int_records_array, RECORDSPERBLOCK);
00322 }
00323
00328 int binarySearch(const string arr[], string x, int n)
00329 {
00330     int int_arr[n];
00331     int int_string;
00332
00333     //convert the records (array of strings) to array of int
00334     for (int i = 0; i < n; i++)
00335     {
00336     if(arr[i] != null_str)
00337           int_arr[i] = stoi(arr[i]);
00338     }
00339
00340     //convert string to find to int
00341     int_string = stoi(x);
00342
00343     int l = 0 ;
00344     int r = n - 1;
00345     while (l <= r)
00346     {
00347       int m = l + (r - l) / 2;
00348           if(DEBUG) {cout « "mid: " « m «endl;}
00349
00350       if(DEBUG) {cout « "comparing " « int_string « " and " « int_arr[m] «endl;}
00351
00352         if ( int_arr[m] == int_string ){
00353         if(DEBUG) {cout « "record found" «endl;}
00354         return m;
00355           }
00356
00357       // If x is greater, ignore left half
00358       if ( int_arr[m] < int_string ){
00359         l = m + 1;
00360       if(DEBUG) {cout « "new l: " « l «endl;}
00361       }
00362
00363         // If x is smaller, ignore right half
00364       else{
00365         r = m - 1;
00366         if(DEBUG) {cout « "new r: " « l «endl;}
00367       }
```

```
00368     }
00369
00370     return -1;
00371 }
00372
00377 void convertStrArrToIntArr(const string strArr[], int intArr[], int ArrLength)
00378 {
00379     //convert the records (array of strings) to array of int
00380     for (int i = 0; i < ArrLength; i++)
00381     {
00382     if(strArr[i] == null_str) //if the record is null
00383       intArr[i] = NULL_INT;
00384     else
00385       intArr[i] = stoi(strArr[i]);
00386     }
00387 }
00388
00393 void convertIntArrToStrArr(string strArr[], int intArr[], int ArrLength)
00394 {
00395   //convert the records (array of strings) to array of int
00396     for (int i = 0; i < ArrLength; i++)
00397     {
00398     if(intArr[i] == NULL_INT)//if the record is null
00399       strArr[i] = null_str;
00400     else
00401       strArr[i] = to_string(intArr[i]);
00402     }
00403 }
```

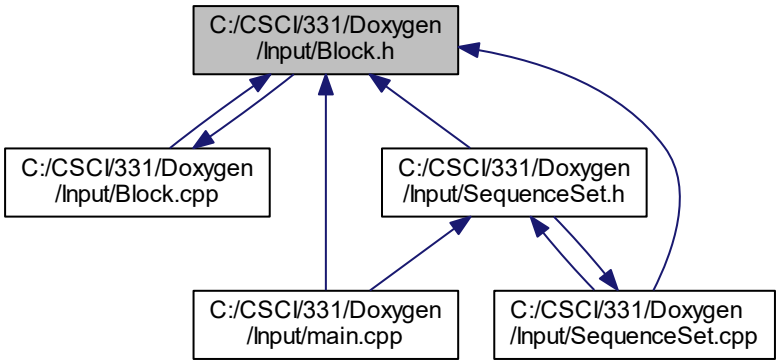
4.3 C:/CSCI/331/Doxygen/Input/Block.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include "Record.h"
#include "Block.cpp"
```

Include dependency graph for Block.h:



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



Classes

- class Block

4.4 Block.h

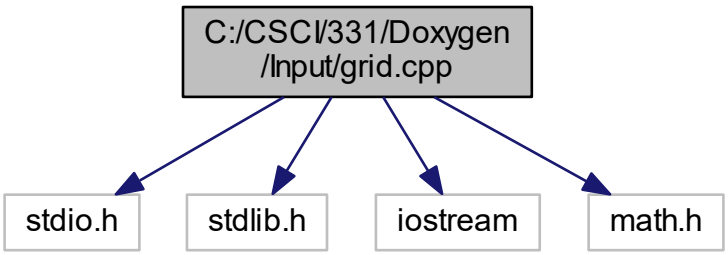
```
00001
00021 #ifndef BLOCK_H
00022 #define BLOCK_H
00023
00024 #include <iostream>
00025 #include <string>
00026 #include <fstream>
00027 #include "Record.h"
00028 using namespace std;
00029
00030 class Block
00031 {
00032   public:
00037     Block();
00038
00043     Block(unsigned long long _RBN);
00044
00049         Block(string[]);
00050
00055     Block(string);
00056
00061         void write(string);
00062
00067     int search(string pKey);
00068
00069     Block * getNextBlock();
00070     Block * getPreviousBlock();
00071     void setNextBlock( Block * nextBlockPtr );
00072     void setPrevBlock( Block * previousBlockPtr );
00073     int getRecordCount();
00074     int getLastRecordPKey();
00080     bool deleteRecord(string pKey);
00081
00086     bool addRecord(string pKey);
00087
00092     void getRecords(Record block[]);
00093
00094         string blockData();
00095         unsigned long long getRBN();
00101     void setRBN(unsigned long long);
00102   private:
00103     void sortRecord();
00104         bool isEmpty;
00105     unsigned long long relativeBlockNumber;
```

```
00106     int recordCount;
00107     string records[RECORDSPERBLOCK];
00108     Block * nextBlock;
00109     Block * previousBlock;
00110 };
00111
00112 #include "Block.cpp"
00113
00114 #endif
```

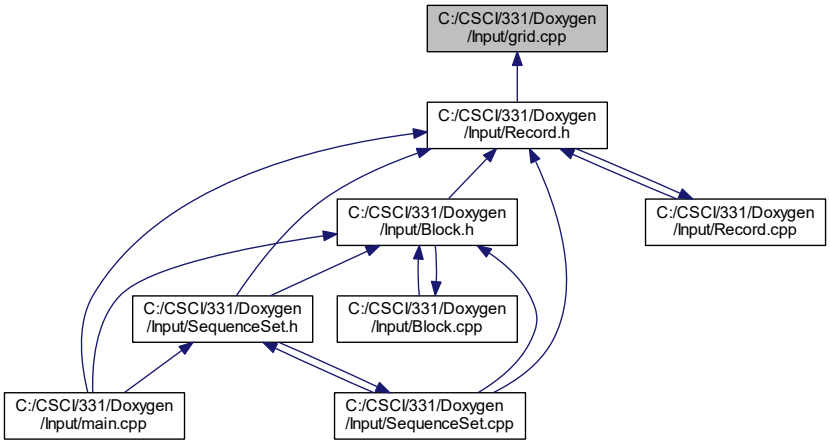
4.5 C:/CSCI/331/Doxygen/Input/grid.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
```

Include dependency graph for grid.cpp:



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



Classes

- class Grid

  Grid class.

4.6 grid.cpp

```
00001
00015 #include <stdio.h>
00016 #include <stdlib.h>
00017 #include <iostream>
00018 #include <math.h>
00019 using namespace std;
00020
00030 class Grid {
00031   private:
00032         float latitude;
00033         float longitude;
00035   public:
00036     Grid();
00037     Grid(float, float);
00038     void setLatitude(float);
00039     void setLongitude(float);
00040     void setLatitude(string);
00041     void setLongitude(string);
00042     float getLatitude();
00043         float getLongitude();
00044         float getDistance(Grid);
00045 };
00046
00051 Grid::Grid(){
00052   latitude = 0;
00053   longitude = 0;
00054 }
00055
00060 Grid::Grid(float _latitude, float _longitude){
00061     latitude = _latitude;
00062     longitude = _longitude;
00063 }
00064
00069 void Grid::setLatitude(float _latitude){
00070     latitude = _latitude;
00071 }
00072
00077 void Grid::setLatitude(string _latitude){
00078     setLatitude(stof(_latitude));
00079 }
00080
00085 void Grid::setLongitude(float _longitude){
00086     longitude = _longitude;
00087 }
00088
00093 void Grid::setLongitude(string _longitude){
00094     setLongitude(stof(_longitude));
00095 }
00096
00101 float Grid::getLatitude(){
00102   return latitude;
00103 }
00104
00109 float Grid::getLongitude(){
00110     return longitude;
00111 }
00112
00117 float Grid::getDistance(Grid _grid){
00118   float distance = pow(latitude - _grid.getLatitude(),2) + pow(longitude - _grid.getLongitude(),2);
00119   distance = sqrt(distance);
00120   return distance;
00121 }
```

4.7 C:/CSCI/331/Doxygen/Input/Header.cpp File Reference

Variables

- const bool DEBUG = false

Set true for debugging output

- const int RECORDSPERBLOCK = 4

  Maximum records for the block

- const int ZIPLENGTH = 6

  Max length of the zip code in digits.
- const int RBNLENGTH = 8

  Max length of the RBN code in digits.
- const int BLOCKLENGTH = RBNLENGTH + RECORDSPERBLOCK ∗ ZIPLENGTH

  Maximum length for the block

- const double FILLPERCENT = 75

  Max length of the RBN code in digits.
- const int BLOCKFILLCOUNT = RECORDSPERBLOCK ∗ (FILLPERCENT/100)

  Max length of the RBN code in digits.
- const string HEADERENDSTRING = "1234567890123456789012345678901234567890123456789012345678901234567890

  String at the end of the header.
- const string DATAFILENAME = "us_postal_codes.txt"

  Data file name.

4.7.1 Variable Documentation

4.7.1.1 DEBUG

```
const bool DEBUG = false
```

Set true for debugging output

4.7.1.2 RECORDSPERBLOCK

```
const int RECORDSPERBLOCK = 4
```

Maximum records for the block

4.7.1.3 ZIPLENGTH

```
const int ZIPLENGTH = 6
```

Max length of the zip code in digits.

4.7.1.4 RBNLENGTH

```
const int RBNLENGTH = 8
```

Max length of the RBN code in digits.

4.7.1.5 BLOCKLENGTH

```
const int BLOCKLENGTH = RBNLENGTH + RECORDSPERBLOCK * ZIPLENGTH
```

Maximum length for the block

4.7.1.6 FILLPERCENT

```
const double FILLPERCENT = 75
```

Max length of the RBN code in digits.

4.7.1.7 BLOCKFILLCOUNT

```
const int BLOCKFILLCOUNT = RECORDSPERBLOCK * (FILLPERCENT/100)
```

Max length of the RBN code in digits.

4.7.1.8 HEADERENDSTRING

```
const string HEADERENDSTRING = "12345678901234567890123456789012345678901234567890123456789012345678901234567
```

String at the end of the header.

4.7.1.9 DATAFILENAME

```
const string DATAFILENAME = "us_postal_codes.txt"
```

Data file name.

4.8 Header.cpp

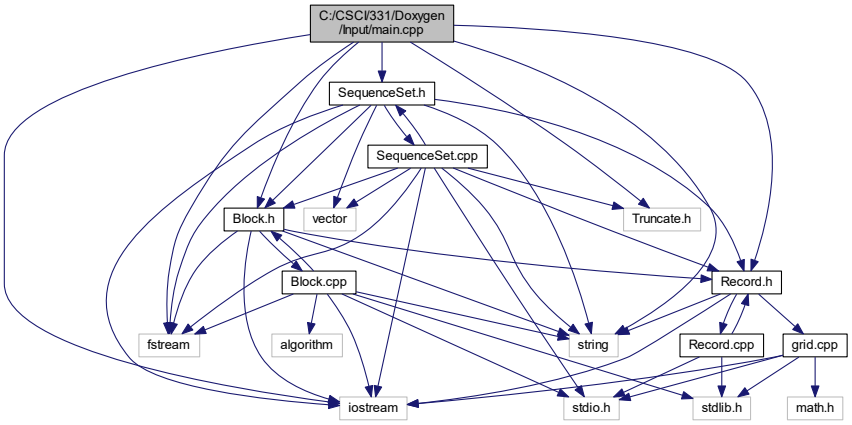
```
00001 extern const bool DEBUG                = false;
00002 extern const int RECORDSPERBLOCK = 4;
00003 extern const int ZIPLENGTH          = 6;
00004 extern const int RBNLENGTH          = 8;
00005 extern const int BLOCKLENGTH          = RBNLENGTH + RECORDSPERBLOCK * ZIPLENGTH;
00006 extern const double FILLPERCENT       = 75;
00007 extern const int BLOCKFILLCOUNT        = RECORDSPERBLOCK * (FILLPERCENT/100);
00008 extern const string HEADERENDSTRING =
       "12345678901234567890123456789012345678901234567890123456789012345678901234567890";

00009 extern const string DATAFILENAME    = "us_postal_codes.txt";
```

4.9 C:/CSCI/331/Doxygen/Input/main.cpp File Reference

```
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include "SequenceSet.h"
#include <string>
#include <fstream>
```

Include dependency graph for main.cpp:



Functions

- void truncateTester ()

  Tests the Truncate Class.
- void recordTester ()
- void blockTester ()
- void nullblockTester ()
- void SSDeleteAndAddRecordTester ()
- int main_menu ()
- void addNewRecord ()
- void searchForRecord ()
- void deleteRecord ()
- void quitProgram ()
- void extremeCoord ()
- int main ()

Variables

- SequenceSet SSClass
- bool quit = false

4.9.1 Function Documentation

4.9.1.1 truncateTester()

```
void truncateTester ( )
```

Tests the Truncate Class.

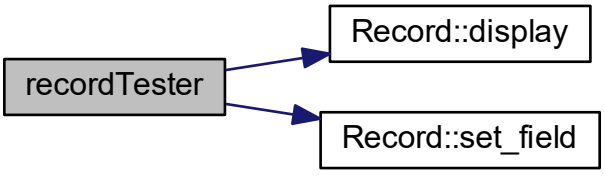
Definition at line 399 of file main.cpp.

4.9.1.2 recordTester()

```
void recordTester ( )
```

Definition at line 410 of file main.cpp.

Here is the call graph for this function:

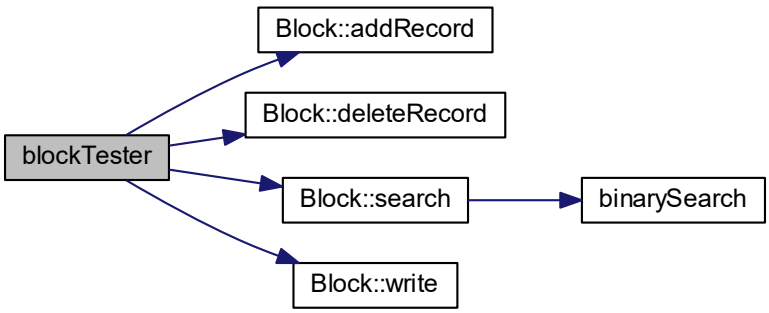


4.9.1.3 blockTester()

```
void blockTester ( )
```

Definition at line 367 of file main.cpp.

Here is the call graph for this function:

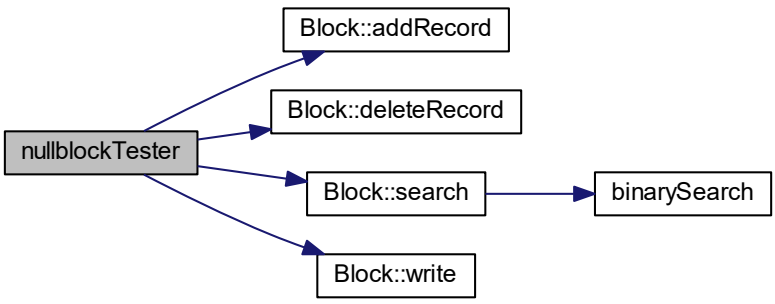


4.9.1.4 nullblockTester()

```
void nullblockTester ( )
```

Definition at line 334 of file main.cpp.

Here is the call graph for this function:

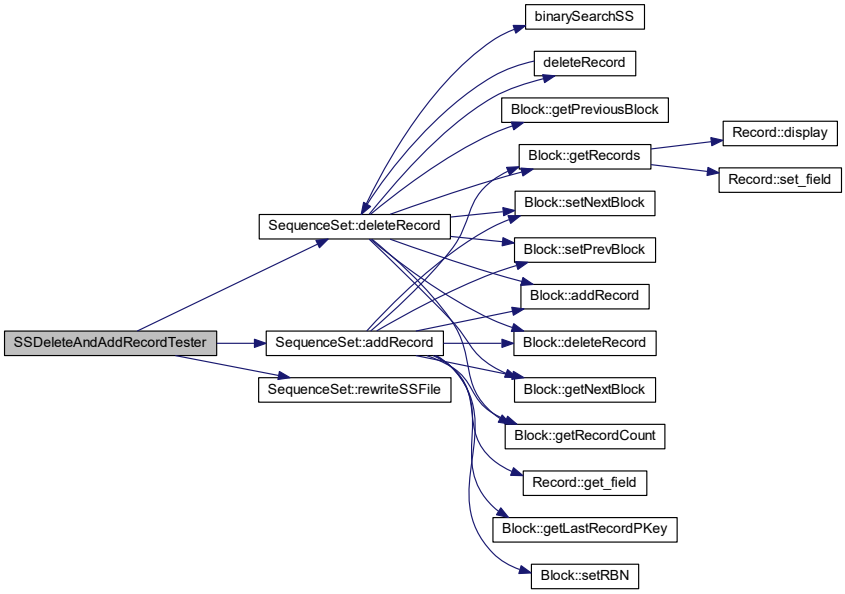


4.9.1.5 SSDeleteAndAddRecordTester()

```
void SSDeleteAndAddRecordTester ( )
```

Definition at line 294 of file main.cpp.

Here is the call graph for this function:

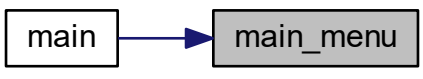


4.9.1.6 main_menu()

```
int main_menu ( )
```

Definition at line 274 of file main.cpp.

Here is the caller graph for this function:

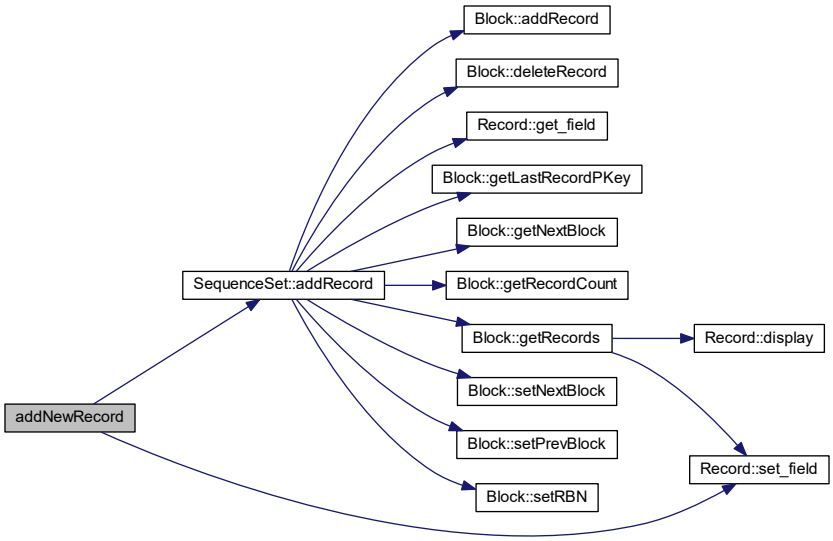


4.9.1.7 addNewRecord()

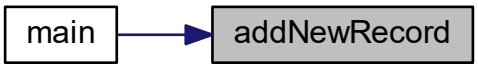
```
void addNewRecord ( )
```

Definition at line 163 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

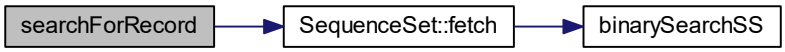


4.9.1.8 searchForRecord()

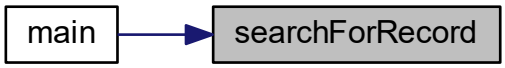
```
void searchForRecord ( )
```

Definition at line 129 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

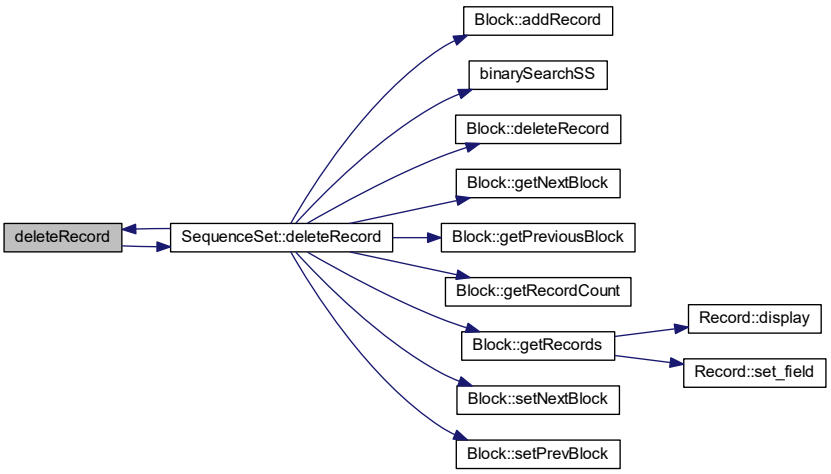


4.9.1.9 deleteRecord()

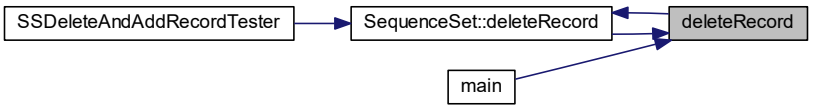
```
void deleteRecord ( )
```

Definition at line 90 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

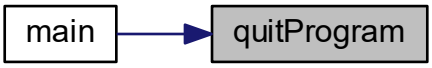


4.9.1.10 quitProgram()

```
void quitProgram ( )
```

Definition at line 85 of file main.cpp.

Here is the caller graph for this function:

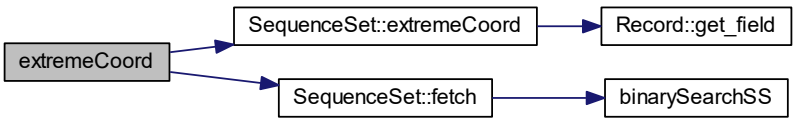


4.9.1.11 extremeCoord()

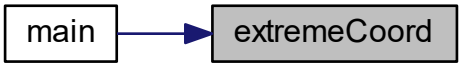
```
void extremeCoord ( )
```

Definition at line 56 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

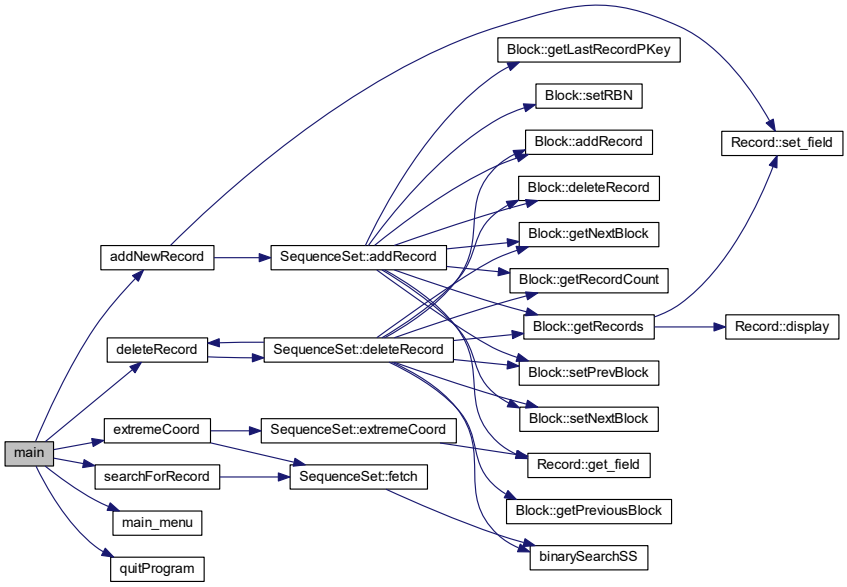


4.9.1.12 main()

```
int main ( )
```

Definition at line 26 of file main.cpp.

Here is the call graph for this function:



4.9.2 Variable Documentation

4.9.2.1 SSClass

```
SequenceSet SSClass
```

Definition at line 23 of file main.cpp.

4.9.2.2 quit

```
bool quit = false
```

Definition at line 24 of file main.cpp.

4.10 main.cpp

```
00001 #include <iostream>
00002 #include "Truncate.h"
00003 #include "Record.h"
00004 #include "Block.h"
00005 #include "SequenceSet.h"
00006 #include <string>
00007 #include <fstream>
00008
00009 using namespace std;
00010
00011 void truncateTester();
00012 void recordTester();
00013 void blockTester();
00014 void nullblockTester();
00015 void SSDeleteAndAddRecordTester();
00016 int main_menu();
00017 void addNewRecord();
00018 void searchForRecord();
00019 void deleteRecord();
00020 void quitProgram();
00021 void extremeCoord();
00022
00023 SequenceSet SSClass;
00024 bool quit = false;
00025
00026 int main(){
00027
00028   int choice;
00029   cout « "Sequence Set Created." « endl;
00030
00031   while( !quit )
00032   {
00033     cout « endl « endl;
00034
00035     choice = main_menu();
00036
00037     switch( choice )
00038     {
00039         case 1: addNewRecord();
00040                 break;
00041         case 2: searchForRecord();
00042                 break;
00043         case 3: deleteRecord();
00044                 break;
00045         case 4: quitProgram();
00046                 break;
00047         case 5: extremeCoord();
00048                 break;
00049         default: cout « "Selecting menu option canceled." « endl;
00050     }
00051   }
00052
00053   return 0;
00054 }
00055
00056 void extremeCoord()
00057 {
00058   string state, dir;
00059   char direction;
00060   cout « "Please enter the state you wish to search in its two letter code with CAPS.\n\t";
00061   cin » state;
00062   cout  « "Please enter the direction you wish to find the extreme with a "
00063         «"character by choosing n, s, e, or w.\n\t";
00064   cin » direction;
00065   switch(direction){
00066   case 'n':
00067     dir = "nothern";
00068     break;
00069   case 'e':
00070     dir = "eastern";
00071     break;
00072   case 's':
```

```
00073     dir = "southern";
00074     break;
00075   case 'w':
00076     dir = "western";
00077     break;
00078   default:
00079     dir = "ERROR";
00080   }
00081   cout  « "The details of the "« dir «" most zipcode\n\t"
00082         « SSClass.fetch(SSClass.extremeCoord(state, direction));
00083 }
00084
00085 void quitProgram()
00086 {
00087     quit = true;
00088 }
00089
00090 void deleteRecord()
00091 {
00092     string field;
00093
00094     //ask user for pKey
00095     while(true)
00096     {
00097         cout « "Please enter the Zip Code of the Record to delete:" « endl;
00098         cin  » field;
00099         if(field.length() > 6 || field.length() < 1)
00100         {
00101             cout « "Invalid Zip Code entered. Please try again." « endl;
00102         }
00103         else
00104         {
00105             break;
00106         }
00107     }
00108
00109     cout « endl « "Searching for Record" « endl;
00110     int position = SSClass.binarySearchSS(field);
00111     if(position != -1)
00112     {
00113         cout « "Deleting Record from Sequence Set." « endl;
00114         if ( SSClass.deleteRecord( stoi( field ) ) )
00115         {
00116             cout « "Record deleted Successfully." « endl;
00117         }
00118         else
00119         {
00120             cout « "Error Deleting Record." « endl;
00121         }
00122     }
00123     else
00124     {
00125         cout « "Record not found in Sequence Set" « endl;
00126     }
00127 }
00128
00129 void searchForRecord()
00130 {
00131     string field;
00132
00133     //ask user for pKey
00134     while(true)
00135     {
00136         cout « "Please enter the Zip Code of the Record to find:" « endl;
00137         cin  » field;
00138         if(field.length() > 6 || field.length() < 1)
00139         {
00140             cout « "Invalid Zip Code entered. Please try again." « endl;
00141         }
00142         else
00143         {
00144             break;
00145         }
00146     }
00147
00148     cout « endl « "Searching for Record" « endl;
00149     int position = SSClass.binarySearchSS(field);
00150     if(position != -1)
00151     {
00152         cout « "Record found in Sequence Set." « endl;
00153         cout « "Displaying Record:" « endl « endl;
00154         cout « SSClass.fetch(field) « endl;
00155     }
00156     else
00157     {
00158         cout « "Record not found in Sequence Set" « endl;
00159     }
```

```
00160
00161 }
00162
00163 void addNewRecord()
00164 {
00165     string field;
00166     Record record;
00167
00168     //ask user for zip code
00169     while(true)
00170     {
00171         cout « "Please enter the Zip Code of the Record:" « endl;
00172         cin  » field;
00173         if(field.length() > 6 || field.length() < 1)
00174         {
00175             cout « "Invalid Zip Code entered. Please try again." « endl;
00176         }
00177         else
00178         {
00179             record.set_field("zip", field);
00180             break;
00181         }
00182     }
00183
00184     //ask user for city
00185     while(true)
00186     {
00187         cout « "Please enter the City of the Record: (use underscore _ as space)" « endl;
00188         cin  » field;
00189         if(field.length() > 31 || field.length() < 1)
00190         {
00191             cout « "Invalid City entered. Please try again." « endl;
00192         }
00193         else
00194         {
00195             for(int i = 0; i < field.length(); i++)
00196             {
00197                 if(field[i] == '_')
00198                     field[i] = ' ';
00199             }
00200             record.set_field("city", field);
00201             break;
00202         }
00203     }
00204
00205     //ask user for state
00206     while(true)
00207     {
00208         cout « "Please enter the State of the Record (two character format: MN):" « endl;
00209         cin  » field;
00210         if( field.length() != 2 )
00211         {
00212             cout « "Invalid State entered. Please try again." « endl;
00213         }
00214         else
00215         {
00216             record.set_field("state", field);
00217             break;
00218         }
00219     }
00220
00221     //ask user for county
00222     while(true)
00223     {
00224         cout « "Please enter the County of the Record:" « endl;
00225         cin  » field;
00226         if(field.length() > 38 || field.length() < 1)
00227         {
00228             cout « "Invalid County entered. Please try again." « endl;
00229         }
00230         else
00231         {
00232             record.set_field("county", field);
00233             break;
00234         }
00235     }
00236
00237     //ask user for longitude
00238     while(true)
00239     {
00240         cout « "Please enter the Longitude of the Record:" « endl;
00241         cin  » field;
00242         if(field.length() > 8 || field.length() < 1)
00243         {
00244             cout « "Invalid Longitude entered. Please try again." « endl;
00245         }
00246         else
```

```
00247         {
00248             record.set_field("long", field);
00249             break;
00250         }
00251     }
00252
00253     //ask user for latitude
00254     while(true)
00255     {
00256         cout « "Please enter the Latitude of the Record:" « endl;
00257         cin  » field;
00258         if(field.length() > 9 || field.length() < 1)
00259         {
00260             cout « "Invalid Latitude entered. Please try again." « endl;
00261         }
00262         else
00263         {
00264             record.set_field("lat", field);
00265             break;
00266         }
00267     }
00268
00269     cout « endl « "New Record Created." « endl;
00270     SSClass.addRecord( record );
00271     cout « "New Record added to Sequence Set." « endl;
00272 }
00273
00274 int main_menu()
00275 {
00276   int userResponce;
00277   while(true)
00278   {
00279       cout « "Please select an option:" « endl;
00280       cout « "1. Add a new Record" « endl;
00281       cout « "2. Search for and Display a Record" « endl;
00282       cout « "3. Delete a Record" « endl;
00283       cout « "4. Quit Program" « endl;
00284       cout « "5. Find the X-treme coordinate of a state" « endl;
00285       cin  » userResponce;
00286
00287       if(userResponce < 1 || userResponce > 5)
00288           cout « "Please enter a valid option" « endl;
00289       else
00290         return userResponce;
00291   }
00292 }
00293
00294 void SSDeleteAndAddRecordTester()
00295 {
00296   SequenceSet SSClass;
00297
00298   SSClass.deleteRecord(1008);
00299   SSClass.deleteRecord(1003);
00300   SSClass.deleteRecord(1004);
00301
00302   string zip = "563";
00303   string place = "Little Falls";
00304   string state = "MN";
00305   string county = "Morrison";
00306   string longitude = "-74.25";
00307   string latitude = "79.72";
00308   Record testRecord(zip, place, state, county, longitude, latitude);
00309   SSClass.addRecord(testRecord);
00310
00311   zip = "1024";
00312   Record testRecord2(zip, place, state, county, longitude, latitude);
00313   SSClass.addRecord(testRecord2);
00314
00315   zip = "1025";
00316   Record testRecord3(zip, place, state, county, longitude, latitude);
00317   SSClass.addRecord(testRecord3);
00318
00319   zip = "1051";
00320   Record testRecord4(zip, place, state, county, longitude, latitude);
00321   SSClass.addRecord(testRecord4);
00322
00323   zip = "1052";
00324   Record testRecord5(zip, place, state, county, longitude, latitude);
00325   SSClass.addRecord(testRecord5);
00326
00327   zip = "300";
00328   Record testRecord6(zip, place, state, county, longitude, latitude);
00329   SSClass.addRecord(testRecord6);
00330
00331   SSClass.rewriteSSFile();
00332 }
00333
```

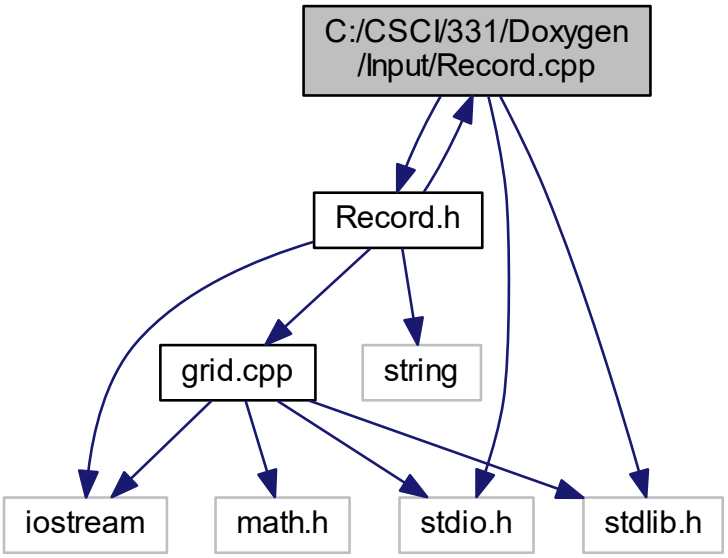
```
00334 void nullblockTester(){
00335   Block * aBlock;
00336   ofstream sequenceSetFile;
00337   string fileName = "Sequence_Set.txt";
00338   sequenceSetFile.open(fileName);
00339   sequenceSetFile « "Hello File\n";
00340   sequenceSetFile.close();
00341 
00342   string records[4] = {"501", "544", "1001", ""};
00343   string blockInfo = "   501   544  1001  1002";
00344 
00345   //test block constructor
00346   Block anotherBlock(blockInfo);
00347   anotherBlock.write(fileName);
00348 
00349   //test block search method
00350   string recordTest = "1002";
00351   aBlock = new Block(1);
00352   cout « "Return 1 if the record was found: " « aBlock->search( recordTest ) « endl;
00353 
00354   recordTest = "103";
00355   aBlock->addRecord(recordTest);
00356 
00357   recordTest = "103";
00358   aBlock->addRecord(recordTest);
00359 
00360   recordTest = "544";
00361   aBlock->deleteRecord(recordTest);
00362 
00363   recordTest = "514";
00364   aBlock->deleteRecord(recordTest);
00365 }
00366 
00367 void blockTester(){
00368   Block aBlock;
00369   ofstream sequenceSetFile;
00370   string fileName = "Sequence_Set.txt";
00371   sequenceSetFile.open(fileName);
00372   sequenceSetFile « "Hello File\n";
00373   sequenceSetFile.close();
00374 
00375   string records[4] = {"501", "544", "1001", ""};
00376   string blockInfo = "   501   544  1001  1002";
00377 
00378   //test block constructor
00379   Block anotherBlock(blockInfo);
00380   anotherBlock.write(fileName);
00381 
00382   //test block search method
00383   string recordTest = "1002";
00384   cout « "Return 1 if the record was found: " « anotherBlock.search( recordTest ) « endl;
00385 
00386   recordTest = "103";
00387   anotherBlock.addRecord(recordTest);
00388 
00389   recordTest = "103";
00390   anotherBlock.addRecord(recordTest);
00391 
00392   recordTest = "544";
00393   anotherBlock.deleteRecord(recordTest);
00394 
00395   recordTest = "514";
00396   anotherBlock.deleteRecord(recordTest);
00397 }
00398 
00399 void truncateTester(){
00400   Truncate t;
00401   Truncate t2(5);
00402   string str = "123456789AB";
00403 
00404   cout « endl « "The String is " « str;
00405   cout « endl « "The String AS it is modified is " « t.modifyString(str);
00406   cout « endl « "The String IF it was modified is " « t2.truncatedString(str);
00407   cout « endl « "The String is " « str « endl;
00408 }
00409 
00410 void recordTester(){
00411   //test default constructor
00412     Record testRecord;
00413   cout « "Default constructor record (should be empty):";
00414   testRecord.display();
00415   cout « endl;
00416 
00417   //test fill record
00418   string zip = "56345";
00419   string place = "Little Falls";
00420   string state = "Minnesota";
```

```
00421   string county = "Morrison";
00422   string longitude = "-74.25";
00423   string latitude = "79.72";
00424
00425   cout « "Fill Record with : " « zip « " " « place « " " « state « " " « county « " " « longitude « "
       " « latitude;
00426
00427   testRecord.set_field( "z", zip );
00428   testRecord.set_field( "place", place );
00429   testRecord.set_field( "STATE", state );
00430   testRecord.set_field( "c", county );
00431   testRecord.set_field( "long", longitude );
00432   testRecord.set_field( "lat", latitude );
00433
00434   testRecord.display();
00435   cout « endl;
00436
00437   //test constructor 2
00438   float longitude_float = 74.25;
00439   float latitude_float = 79.72;
00440
00441   Record testRecord2(zip, place, state, county, longitude, latitude);
00442
00443   cout « "Constructor2 record (record should be full):";
00444   testRecord2.display();
00445
00446     //test constructor 3
00447   Grid grid_test(longitude_float, latitude_float);
00448
00449   Record testRecord3(zip, place, state, county, grid_test);
00450
00451   cout « "Constructor3 record (record should be full):";
00452   testRecord3.display();
00453
00454   //test display field
00455   cout « endl « "Test Display Field, display city:";
00456   testRecord3.display("CITY");
00457   cout « " expected: Little Falls" « endl;
00458
00459   cout « "Test Display Field, display state:";
00460   testRecord3.display("STATE");
00461   cout « " expected: Minnesota" « endl;
00462 }
```

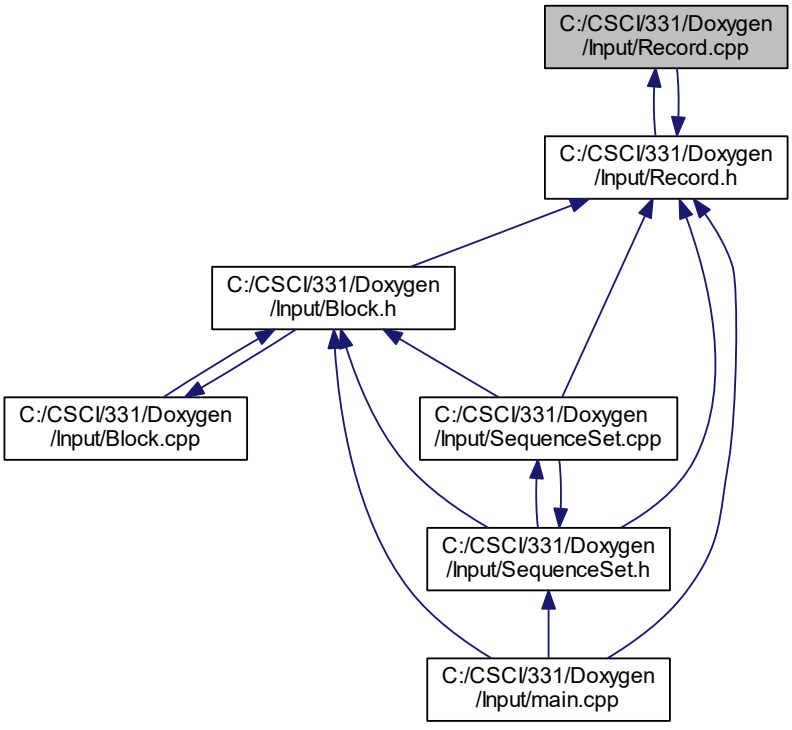
4.11 C:/CSCI/331/Doxygen/Input/Record.cpp File Reference

```
#include "Record.h"
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for Record.cpp:



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



4.12 Record.cpp

```
00001
00017 #include "Record.h"
00018 #include <stdio.h>
00019 #include <stdlib.h>
00020 using namespace std;
00021
00022 Record::Record()
00023 {
00024   zip_code = "";
00025   place_name = "";
00026   state = "";
00027   county = "";
00028   this -> set_longitude_latitude( 0.0, 0.0 );
00029   if(DEBUG) {cout « "Made an empty record.\n";}
00030 }
00031
00032 Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid _gridPoint)
00033 {
00034   zip_code = _zip_code;
00035   place_name = _place_name;
00036   state = _state;
00037   county = _county;
00038   this -> set_grid_point( _gridPoint );
00039   if(DEBUG) {cout « "Made a filled record using a gridPoint.\n";}
00040 }
00041
00042 Record::Record(string _zip_code, string _place_name, string _state, string _county, string latitude,
       string longitude)
00043 {
00044   float lon;
00045   float lat;
00046
00047   try{
00048     lon = string_to_float( longitude );
00049   }
00050   catch(...){
00051     cout « "ERROR SETTING LONGITUDE, SETTING IT TO 0\n";
00052     lon = 0;
00053   }
00054
00055   try{
00056     lat = string_to_float( latitude );
00057   }
00058   catch(...){
00059     cout « "ERROR SETTING LATITUDE IN " « zip_code « ", SETTING IT TO 0\n";
00060     lat = 0;
00061   }
00062
00063   zip_code = _zip_code;
00064   place_name = _place_name;
00065   state = _state;
00066   county = _county;
00067   this -> set_longitude_latitude( lon, lat );
00068   if(DEBUG) {cout « "Made a filled record using string lat/longs.\n";}
00069 }
00070
00071 void Record::display()
00072 {
00073     if(DEBUG) {cout « "Displaying the whole record from the record.\n";}
00074     cout « endl
00075        « "Zipcode:\t" « get_field("Zip")
00076        « "\nPlace:\t\t" « get_field("City")
00077        « "\nState:\t\t" « get_field("State")
00078        « "\nCounty:\t\t" « get_field("County")
00079        « "\nLongitude:\t" « get_field("Longitude")
00080        « "\nLatitude:\t" « get_field("Latitude")
00081        « endl;
00082 }
00083
00084 void Record::display(string field)
00085 {
00086   if(DEBUG) {cout « "Displaying the "« field «" portion of the record.\t";}
00087   for(int i = 0; field[i] != NULL; i++){
00088       field[i] = toupper(field[i]);
00089   }
00090
00091   if(field=="Z" || field=="ZIP")
00092     cout « zip_code « endl;
00093   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00094     cout « place_name « endl;
00095   else if(field=="STATE")
00096     cout « state « endl;
00097   else if(field=="COUNTY")
00098     cout « county « endl;
00099   else if(field=="G" || field=="GRID")
```

```
00100     cout « gridPoint.getLatitude() « " " « gridPoint.getLongitude() « endl;
00101   else if(field == "LAT" || field == "LATITUDE")
00102     cout « gridPoint.getLatitude() « endl;
00103   else if(field == "LONG" || field == "LONGITUDE")
00104     cout « gridPoint.getLongitude() « endl;
00105   else
00106     cout « "Invalid field has been entered." « endl;
00107 }
00108
00109 string Record::get_field(string field)
00110 {
00111   if(DEBUG) {cout « "Retrieving the "« field «" portion of the record.\t";}
00112   string returnString;
00113   for(int i = 0; field[i] != NULL; i++){
00114       field[i] = toupper(field[i]);
00115   }
00116
00117   if(field=="Z" || field=="ZIP")
00118     returnString = zip_code;
00119   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00120     returnString = place_name;
00121   else if(field=="STATE")
00122     returnString = state;
00123   else if(field=="COUNTY")
00124     returnString = county;
00125   else if(field=="G" || field=="GRID")
00126     returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
00127   else if(field == "LAT" || field == "LATITUDE")
00128     returnString = to_string(gridPoint.getLatitude());
00129   else if(field == "LONG" || field == "LONGITUDE")
00130     returnString = to_string(gridPoint.getLongitude());
00131   else
00132     returnString = "ERROR";
00133
00134   return returnString;
00135 }
00136
00137 void Record::set_field(string field, string data)
00138 {
00139   if(DEBUG) {cout « "Setting the "« field «" portion of the record from "« get_field(field) « " to"«
       data «".\n";}
00140   for(int i = 0; field[i] != NULL; i++){
00141       field[i] = toupper(field[i]);
00142   }
00143
00144   for(int i = 0; data[i] != NULL; i++){
00145       data[i] = toupper(data[i]);
00146   }
00147
00148   if(field=="Z" || field=="ZIP")
00149     zip_code = data;
00150   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00151     place_name = data;
00152   else if(field=="STATE")
00153     state = data;
00154   else if(field=="COUNTY")
00155     county = data;
00156   else if(field=="G" || field=="GRID")
00157     cout « "grid setter needs to implemented";
00158   else if(field == "LAT" || field == "LATITUDE")
00159     gridPoint.setLatitude(data);
00160   else if(field == "LONG" || field == "LONGITUDE")
00161     gridPoint.setLongitude(data);
00162   else
00163     cout « "ERROR" « endl;
00164 }
00165
00166 void Record::set_longitude_latitude(float longitude, float latitude)
00167 {
00168   gridPoint.setLatitude( latitude );
00169   gridPoint.setLongitude( longitude );
00170 }
00171
00172 void Record::set_grid_point(Grid _gridPoint)
00173 {
00174   gridPoint.setLatitude( _gridPoint.getLatitude() );
00175   gridPoint.setLongitude ( _gridPoint.getLongitude() );
00176 }
00177
00178 //helper functions
00179
00180 float Record::string_to_float(string str)
00181 {
00182   size_t size;
00183   float float_value = stof(str, &size);
00184
00185   return float_value;
```

```
00186 }
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197 // /**---------------------------------------------------------------------------
00198 //  * @Record.cpp
00199 //  * Class Record (Contains information about individual zipcodes)
00200 //  * @author Tyler Lahr, Ryan Sweeney, and Seth Pomahatch
00201 //  * (Additional comments by Mark Christenson)
00202 //  *---------------------------------------------------------------------------
00203 //  * Record class:  Used by Sequence Set Class
00204 //  *   includes additional features:
00205 //  *    -- Display the whole record it represents
00206 //  *    -- Display a field with in the record
00207 //  *    -- Return a field as a string
00208 //  *    -- Return the latitude
00209 //  *    -- Return the longitude
00210 //  *---------------------------------------------------------------------------
00211 //  */
00212
00213 // #include "Record.h"
00214 // #include <stdio.h>
00215 // #include <stdlib.h>
00216 // using namespace std;
00217
00218 // Record::Record()
00219 // {
00220 //   zip_code = "";
00221 //   place_name = "";
00222 //   state = "";
00223 //   county = "";
00224 //   this -> set_longitude_latitude( 0.0, 0.0 );
00225 //   if(DEBUG) {cout « "Made an empty record.\n";}
00226 // }
00227
00228 // Record::Record(string _zip_code, string _place_name, string _state, string _county, Grid
       _gridPoint)
00229 // {
00230 //   zip_code = _zip_code;
00231 //   place_name = _place_name;
00232 //   state = _state;
00233 //   county = _county;
00234 //   this -> set_grid_point( _gridPoint );
00235 //   if(DEBUG) {cout « "Made a filled record using a gridPoint.\n";}
00236 // }
00237
00238 // Record::Record(string _zip_code, string _place_name, string _state, string _county, string
       latitude, string longitude)
00239 // {
00240 //   float lon = string_to_float( longitude );
00241 //   float lat = string_to_float( latitude );
00242
00243 //   zip_code = _zip_code;
00244 //   place_name = _place_name;
00245 //   state = _state;
00246 //   county = _county;
00247 //   this -> set_longitude_latitude( lon, lat );
00248 //   if(DEBUG) {cout « "Made a filled record using string lat/longs.\n";}
00249 // }
00250
00251 // void Record::display()
00252 // {
00253 //   if(DEBUG) {cout « "Displaying the whole record from the record.\n";}
00254 //   cout « endl
00255 //       « "Zipcode: " « get_field("Zip")
00256 //       « " Place: " « get_field("Place_name")
00257 //        « " State: " « get_field("State")
00258 //        « " County: " « get_field("County")
00259 //        « " Longitude: " « get_field("Longitude")
00260 //        « " Latitude: " « get_field("Latitude")
00261 //        « endl;
00262 // }
00263
00264 // void Record::display(string field)
00265 // {
00266 //   if(DEBUG) {cout « "Displaying the "« field «" portion of the record.\n";}
00267 //   for(int i = 0; field[i] != NULL; i++){
00268 //     field[i] = toupper(field[i]);
00269 //   }
00270
```

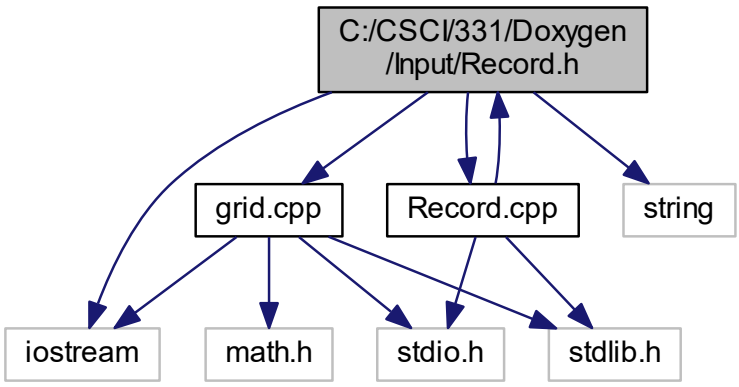
```
00271 //   if(field=="Z" || field=="ZIP")
00272 //     cout « zip_code « endl;
00273 //   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00274 //     cout « place_name « endl;
00275 //   else if(field=="STATE")
00276 //     cout « state « endl;
00277 //   else if(field=="COUNTY")
00278 //     cout « county « endl;
00279 //   else if(field=="G" || field=="GRID")
00280 //     cout « gridPoint.getLatitude() « " " « gridPoint.getLongitude() « endl;
00281 //   else if(field == "LAT" || field == "LATITUDE")
00282 //     cout « gridPoint.getLatitude() « endl;
00283 //   else if(field == "LONG" || field == "LONGITUDE")
00284 //     cout « gridPoint.getLongitude() « endl;
00285 //   else
00286 //     cout « "Invalid field has been entered." « endl;
00287 // }
00288
00289 // string Record::get_field(string field)
00290 // {
00291 //   if(DEBUG) {cout « "Retrieving the "« field «" portion of the record.\n";}
00292 //   string returnString;
00293 //   for(int i = 0; field[i] != NULL; i++){
00294 //     field[i] = toupper(field[i]);
00295 //   }
00296
00297 //   if(field=="Z" || field=="ZIP")
00298 //     returnString = zip_code;
00299 //   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00300 //     returnString = place_name;
00301 //   else if(field=="STATE")
00302 //     returnString = state;
00303 //   else if(field=="COUNTY")
00304 //     returnString = county;
00305 //   else if(field=="G" || field=="GRID")
00306 //     returnString = to_string(gridPoint.getLatitude()) + " " + to_string(gridPoint.getLongitude());
00307 //   else if(field == "LAT" || field == "LATITUDE")
00308 //     returnString = to_string(gridPoint.getLatitude());
00309 //   else if(field == "LONG" || field == "LONGITUDE")
00310 //     returnString = to_string(gridPoint.getLongitude());
00311 //   else
00312 //     returnString = "ERROR";
00313
00314 //   return returnString;
00315 // }
00316
00317 // void Record::set_field(string field, string data)
00318 // {
00319 //   if(DEBUG) {cout « "Setting the "« field «" portion of the record from "« get_field(field) « "
      to"« data «".\n";}
00320 //   for(int i = 0; field[i] != NULL; i++){
00321 //       field[i] = toupper(field[i]);
00322 //   }
00323
00324 //   for(int i = 0; data[i] != NULL; i++){
00325 //       data[i] = toupper(data[i]);
00326 //   }
00327
00328 //   if(field=="Z" || field=="ZIP")
00329 //     zip_code = data;
00330 //   else if(field=="CITY" || field=="P" || field=="PLACE_NAME")
00331 //     place_name = data;
00332 //   else if(field=="STATE")
00333 //     state = data;
00334 //   else if(field=="COUNTY")
00335 //     county = data;
00336 //   else if(field=="G" || field=="GRID")
00337 //     cout « "grid setter needs to implemented";
00338 //   else if(field == "LAT" || field == "LATITUDE")
00339 //     gridPoint.setLatitude(data);
00340 //   else if(field == "LONG" || field == "LONGITUDE")
00341 //     gridPoint.setLongitude(data);
00342 //   else
00343 //     cout « "ERROR" « endl;
00344 // }
00345
00346 // void Record::set_longitude_latitude(float longitude, float latitude)
00347 // {
00348 //   gridPoint.setLatitude( latitude );
00349 //   gridPoint.setLongitude( longitude );
00350 // }
00351
00352 // void Record::set_grid_point(Grid _gridPoint)
00353 // {
00354 //   gridPoint.setLatitude( _gridPoint.getLatitude() );
00355 //   gridPoint.setLongitude ( _gridPoint.getLongitude() );
00356 // }
```

```
00357
00358 // float Record::string_to_float(string str)
00359 // {
00360 //   size_t size;
00361 //   float float_value = stof(str, &size);
00362
00363 //   return float_value;
00364 // }
00365
00366 // //   if gridPoint.setLatitude(data);
00367 // //   else if(field == "LONG" || field == "LONGITUDE")
00368 // //     gridPoint.setLongitude(data);
00369 // //   else
00370 // //     cout « "ERROR" « endl;
00371 // // }
00372
00373 // void Record::set_longitude_latitude(float longitude, float latitude)
00374 // {
00375 //   gridPoint.setLatitude( latitude );
00376 //   gridPoint.setLongitude( longitude );
00377 // }
00378
00379 // void Record::set_grid_point(Grid _gridPoint)
00380 // {
00381 //   gridPoint.setLatitude( _gridPoint.getLatitude() );
00382 //   gridPoint.setLongitude ( _gridPoint.getLongitude() );
00383 // }
00384
00385 // float Record::string_to_float(string str)
00386 // {
00387 //   size_t size;
00388 //   float float_value = stof(str, &size);
00389
00390 //   return float_value;
00391 // }
```

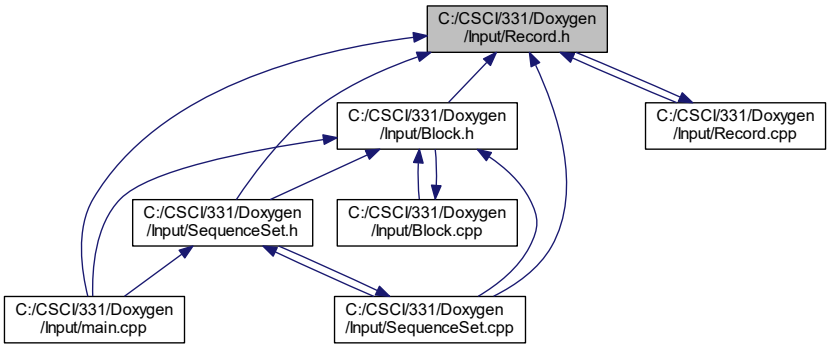
4.13 C:/CSCI/331/Doxygen/Input/Record.h File Reference

```
#include <iostream>
#include <string>
#include "grid.cpp"
#include "Record.cpp"
```

Include dependency graph for Record.h:



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



Classes

- class Record

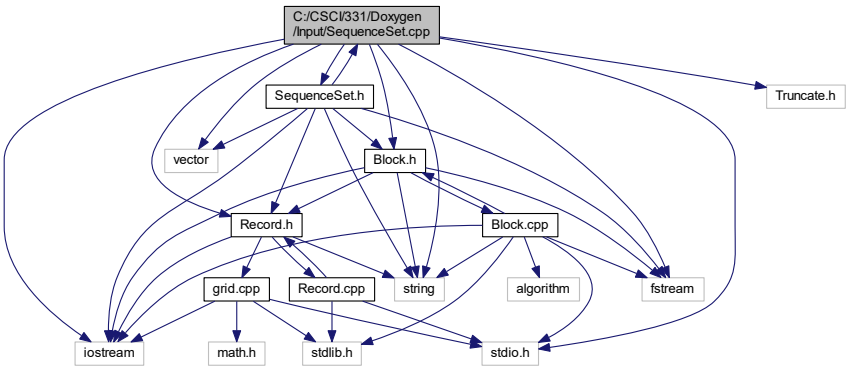
4.14 Record.h

```
00001
00017 #ifndef RECORD_H
00018 #define RECORD_H
00019
00020 #include <iostream>
00021 #include <string>
00022 #include "grid.cpp"
00023 using namespace std;
00024
00025 class Record
00026 {
00027   public:
00032     Record();
00033
00038     Record(string, string, string, string, Grid);
00039
00044     Record(string, string, string, string, string, string);
00045
00050     void display();
00051
00056     void display(string); //This might benefit from calling get_field
00057
00062     string get_field(string); //This should have a switch statement
00063
00069     void set_field(string, string);
00070
00075     void set_longitude_latitude(float, float);
00076
00081     void set_grid_point(Grid);
00082
00083   private:
00084     bool isEmpty;
00085     string zip_code;
00086     string place_name;
00087     string state;
00088     string county;
00089     Grid gridPoint;
00095     float string_to_float(string);
00096 };
00097
00098 #include "Record.cpp"
00099
00100 #endif
```

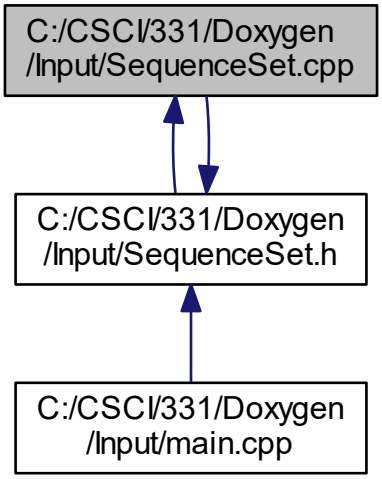
4.15 C:/CSCI/331/Doxygen/Input/SequenceSet.cpp File Reference

```
#include "SequenceSet.h"
#include <iostream>
#include "Truncate.h"
#include "Record.h"
#include "Block.h"
#include <string>
#include <fstream>
#include <vector>
#include <stdio.h>
```

Include dependency graph for SequenceSet.cpp:



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



Functions

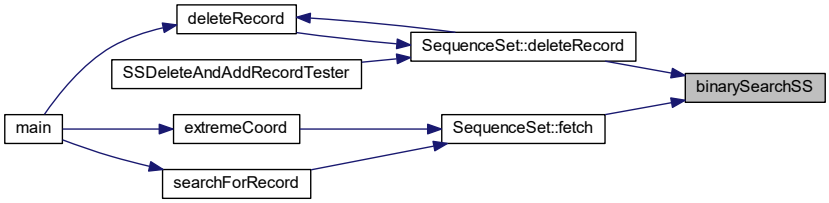
- int binarySearchSS (const string arr[ ], string x, int n)

4.15.1 Function Documentation

4.15.1.1 binarySearchSS()

```
int binarySearchSS (
            const string arr[],
            string x,
            int n )
```

Here is the caller graph for this function:



4.16 SequenceSet.cpp

```
00001
00014 #include "SequenceSet.h"
00015 #include <iostream>
00016 #include "Truncate.h"
00017 #include "Record.h"
00018 #include "Block.h"
00019 #include "SequenceSet.h"
00020 #include <string>
00021 #include <fstream>
00022 #include <vector>
00023 #include <stdio.h>
00024
00025 using namespace std;
00026
00027 //binarySearch recycled from block
00028 int binarySearchSS(const string arr[], string x, int n);
00029
00030 SequenceSet::SequenceSet(){
00031   ofstream SSFile;
00032   SSFile.open(SSFileName);
00033   SSFile « "Sequence Set File\n";
00034   SSFile.close();
00035   recordCount = getRecordCount();
00036   fillIndex();
00037   Block * currentBlock = headBlock;
00038   blockCount = 0;
00039   for(unsigned long long i = 0; i < recordCount; i++){
00040     if(i%BLOCKFILLCOUNT == 0 && i != 0){
00041         if(DEBUG){cout « "Making a new block for the chain." « endl;}
00042       blockCount++;
00043       Block * newBlock = new Block(blockCount);
00044       currentBlock->setNextBlock(newBlock);
00045       newBlock->setPrevBlock(currentBlock);
00046       currentBlock = newBlock;
00047     }
00048     if(DEBUG){cout«"Passing "«to_string(pKeyIndex.at(i))«" into the add function."«endl;}
00049     currentBlock->addRecord(to_string(pKeyIndex.at(i)));
00050   }
00051   writeBlocks();
00052
00053   //reset the record avail list
00054   ofstream recordAvailList;
```

```
00055   recordAvailList.open(recordAvailListFileName);
00056   recordAvailList « "";
00057   recordAvailList.close();
00058   sKeyStateBuilder();
00059 }// End default constructor
00060
00061 unsigned long long SequenceSet::headerLength(string _fileName){
00062   fstream data;
00063   unsigned long long length = 0;
00064   unsigned long long L = 0;
00065   data.open(_fileName);
00066   string str;
00067
00068   if(DEBUG){cout « "String outside while loop, in headerLength: " « str « endl;}
00069   while(data.peek() != EOF){
00070     if(DEBUG && false){cout « "String in headerLength: " « str « endl;}
00071     getline(data, str);
00072     length += str.length();
00073     length++;
00074     if(str == HEADERENDSTRING){
00075       L = length;
00076       if(DEBUG){cout«"L defined: "« L «"\n";}
00077     }
00078   }
00079
00080   data.close();
00081
00082   return L;
00083 }// End headerLength
00084
00085 unsigned int SequenceSet::getRecordCount(){
00086     string fileName = DATAFILENAME;
00087     string field;
00088     string str = "";
00089     char c;
00090     fstream data;
00091     unsigned int recordCount = 0;
00092     data.open(fileName);
00093     getline(data, field); //Skip title
00094     while(data.peek() != ':' ){
00095         data.get(c);
00096         field += c;
00097         if(DEBUG && true){cout « "Char c: " « c « endl;}
00098     }
00099
00100     //This while is to skip non number values before approaching what to do with the values
00101     while(data.peek() < '0' || data.peek() > '9' ){
00102         data.get(c);
00103     }
00104     getline(data, str);
00105
00106     recordCount = stoi(str);
00107
00108     if(DEBUG) {cout « "String: " « str « "\nrecords: " « recordCount « endl;}
00109     if(field == "Records"){
00110         getline(data, str);
00111         recordCount = stoi(str);
00112         if(DEBUG){cout « "Record Count: " « recordCount « endl;}
00113     }
00114     data.close();
00115
00116     return recordCount;
00117 }// End getRecordCount
00118
00119 void SequenceSet::fillIndex(){
00120     string field;
00121     string str = "";
00122     char c;
00123     fstream data;
00124
00125     data.open("RecordOffsets.txt");
00126
00127     for(unsigned int i = 0; i < recordCount; i++){
00128         string recordData = "";
00129         getline(data, recordData);
00130         if(DEBUG){cout « "recordData: " « recordData « endl;}
00131         str = "";
00132         for(int j = 0; j < ZIPLENGTH; j++){
00133           str += recordData[j];
00134         }
00135         //index[i][0] = stoi(str);  //five chars of string
00136         pKeyIndex.push_back(stoi(str));
00137         if(DEBUG){cout « "String: " « str « endl;}
00138         if(DEBUG){cout « "pKeyIndex.at(i): " « pKeyIndex.at(i) «endl;}
00139         str = "";
00140         for(int j = ZIPLENGTH; j < recordData.length(); j++){
00141           str += recordData[j];
```

```
00142         }
00143         //index[i][1] = stoi(str);  //the rest of the string
00144         offsetIndex.push_back(stoi(str));
00145         if(DEBUG){cout « "String: " « str « endl;}
00146         if(DEBUG){cout « "offsetIndex.at("«i«"): " « offsetIndex.at(i) «endl;}
00147     }
00148     data.close();
00149 }// End fillIndex
00150
00151 string SequenceSet::fetch(string pKey){
00152   fstream data;
00153   data.open(DATAFILENAME);
00154   string returnString = "";
00155   for(int i = ZIPLENGTH - pKey.length(); i > 0; i--){
00156     if(DEBUG){cout « "For loop in fetch. i = " « i « endl;}
00157     returnString += " ";
00158   }
00159   returnString = pKey;
00160   returnString += " not found.\n";
00161
00162   int position;
00163   if(pKey != ""){
00164     position = binarySearchSS(pKey);
00165   }
00166   if(DEBUG) {cout « "Searching "« pKey « " returned: " « position « endl;}
00167   if(position>=0 && pKey != ""){
00168     data.seekg(offsetIndex.at(binarySearchSS(pKey)));
00169     getline(data, returnString);
00170   }
00171   data.close();
00172
00173   return returnString;
00174 }// End fetch with string
00175
00176 string SequenceSet::fetch(unsigned int pKey){
00177   return fetch(to_string(pKey));
00178 }// End fetch with int
00179
00180 void SequenceSet::makeRecordOffsets(string fileName){
00181     string zip = "       ";
00182     fstream data;
00183     ofstream index;
00184     string str;
00185     index.open("RecordOffsets.txt");
00186     unsigned long long offset = headerLength(fileName);
00187     data.open(fileName);
00188     data.seekg(offset);
00189
00190     if(DEBUG && false){cout « "String in makeRecordOffsets is: " « str « endl;}
00191     getline(data, str);
00192
00193     while(data.peek()!=EOF){
00194       if (DEBUG && false){cout « str « endl;}
00195     for(int i = 0; i < ZIPLENGTH; i++){
00196       zip[i] = str[i];
00197     }
00198     if(DEBUG && false){cout«zip« " is at " « offset «endl;}
00199     index « zip « offset « endl;
00200     getline(data, str);
00201     offset += str.length();
00202     offset++;
00203     }
00204
00205     data.close();
00206     index.close();
00207 }//End makeRecordOffsets
00208
00209
00210 int SequenceSet::binarySearchSS(string x)
00211 {
00212     //int int_arr[n];
00213     unsigned int n = recordCount;
00214     int int_string;
00215 /*
00216     //convert the records (array of strings) to array of int
00217     for (unsigned int i = 0; i < n; i++)
00218     {
00219     if(arr[i] != null_str)
00220           int_arr[i] = stoi(arr[i]);
00221     }
00222 */
00223     //convert string to find to int
00224     if(DEBUG){cout « "(stoi)ing this string: \"" « x « "\"\n";}
00225     try{
00226     int_string = stoi(x);    unsigned int l = 0 ;
00227     unsigned int r = n - 1;
00228     while (l <= r)
```

```
00229     {
00230       int m = l + (r - l) / 2;
00231           if(DEBUG) {cout « "mid: " « m «endl;}
00232
00233       //if(DEBUG) {cout « "comparing " « int_string « " and " « int_arr[m] «endl;}
00234       if(DEBUG) {cout « "comparing " « int_string « " and " « pKeyIndex.at(m) «endl;}
00235
00236         if ( pKeyIndex.at(m) == int_string ){
00237         if(DEBUG) {cout « "record found" «endl;}
00238         return m;
00239           }
00240
00241       // If x is greater, ignore left half
00242       if ( pKeyIndex.at(m) < int_string ){
00243         l = m + 1;
00244         if(DEBUG) {cout « "new l: " « l «endl;}
00245       }
00246
00247         // If x is smaller, ignore right half
00248       else{
00249         r = m - 1;
00250         if(DEBUG) {cout « "new r: " « l «endl;}
00251       }
00252     }
00253 }
00254     catch(...){cout « "ERROR (stoi)ING THIS STRING: \"" « x « "\"\n";}
00255
00256     return -1;
00257 }// End binarySearchSS
00258
00259 Record SequenceSet::fillRecord(string RecordString){
00260   string zip_code, place_name, state, county, latitude, longitude;
00261   int position = 0;
00262   if(DEBUG){cout  « "In fillRecord for Sequence Set Class\n\tRecordString: "
00263                   « RecordString « endl;}
00264   zip_code = "";
00265   for(auto i = 0; i < ZIPLENGTH    ; i++){
00266     if(RecordString[position] != ' '){
00267       zip_code += RecordString[position];
00268     }
00269     position++;
00270   }
00271
00272   place_name = "";
00273   for(int i = 0; i < 31/*Length of place name*/; i++){
00274     if(RecordString[position] != ' '){
00275       place_name += RecordString[position];
00276     }
00277     position++;
00278   }
00279
00280   state = "";
00281   for(int i = 0; i < 2/*Length of state*/; i++){
00282     if(RecordString[position] != ' '){
00283       state += RecordString[position];
00284     }
00285     position++;
00286   }
00287
00288   county = "";
00289   for(int i = 0; i < 38/*Length of county*/; i++){
00290     if(RecordString[position] != ' '){
00291       county += RecordString[position];
00292     }
00293     position++;
00294   }
00295
00296   latitude = "";
00297   for(int i = 0; i < 9/*Length of latitude*/; i++){
00298     if(RecordString[position] != ' '){
00299       latitude += RecordString[position];
00300     }
00301     position++;
00302   }
00303
00304   longitude = "";
00305   for(int i = 0; i < 8/*Length of longitude*/; i++){
00306     if(RecordString[position] != ' '){
00307       longitude += RecordString[position];
00308     }
00309     position++;
00310   }
00311   if(DEBUG){cout  « "\tRecordElements: " « "\n\t\t"
00312                   « zip_code « "\n\t\t" « place_name « "\n\t\t"
00313                       « state « "\n\t\t" « county « "\n\t\t"
00314                   « latitude « "\n\t\t" « longitude « endl;}
00315
```

```
00316   Record returnRecord(zip_code, place_name, state, county, latitude, longitude);
00317
00318   if(DEBUG){returnRecord.display();}
00319
00320   return returnRecord;
00321 }// End fillRecord
00322
00323 void SequenceSet::writeBlocks(){
00324   Block * currentBlock = headBlock;
00325   for(auto i = 0; i < blockCount; i ++){
00326     if(DEBUG){cout « "Writing block "« i «" from the chain." « endl;}
00327     currentBlock->write(SSFileName);
00328     currentBlock = currentBlock->getNextBlock();
00329   }
00330 }// End writeBlocks
00331
00332 void SequenceSet::fillRecordBlock(unsigned long long blockID){
00333   string str, zip, passed;
00334   Block * currentBlock = headBlock;
00335   for(auto i = 0; i < blockID; i++){
00336     currentBlock = currentBlock->getNextBlock();
00337   }
00338
00339   currentBlock->getRecords(recordBlock);
00340   for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00341     passed = fetch(recordBlock[i].get_field("ZIP"));
00342     if(DEBUG){
00343       cout  « "\n*****************************************"
00344             « "\nString passed to fill record: " « passed « endl;
00345     }
00346     if(passed != " not found.\n" && passed != " not found."){
00347       recordBlock[i] = fillRecord(passed);
00348       if(DEBUG){recordBlock[i].display();}
00349     }
00350   }
00351 }// End fillRecordBlock
00352
00353 void SequenceSet::addBlockStateKey(unsigned long long blockID){
00354   fillRecordBlock(blockID);
00355   Block * currentBlock = headBlock;
00356
00357   for(auto i = 0; i < blockID; i++){
00358     currentBlock = currentBlock->getNextBlock();
00359   }
00360
00361   for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00362     string state = recordBlock[i].get_field("state");
00363
00364     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00365       string state = recordBlock[i].get_field("state");
00366
00367       if(state != ""){
00368     bool stateFound = false;
00369     unsigned int index = 0;
00370
00371     if(stateZips.size() == 0){
00372             vector <string> newRow;
00373             newRow.push_back(state);
00374             stateZips.push_back(newRow);
00375     }
00376
00377     while(index < stateZips.size() && !stateFound){
00378         if(stateZips[index].at(0) == state){
00379             if(DEBUG){cout « "Found " « state « " at index = " « index « endl;}
00380             stateFound = true;
00381         }
00382         else{index++;}
00383     }
00384
00385     if(!stateFound){
00386         if(DEBUG){cout « state«" not found.\n";}
00387             vector <string> newRow;
00388             newRow.push_back(state);
00389             stateZips.push_back(newRow);
00390         if(DEBUG){cout « stateZips[index].at(0)«" pushed successfully.\n";}
00391         if(DEBUG){
00392             stateZips[index].push_back(":)");
00393             cout « "Pushing a smily :)\n";
00394             cout « stateZips[index].at(1) « endl;
00395             stateZips[index].pop_back();
00396         }
00397     }
00398
00399     if(DEBUG){cout « "Pushing " « recordBlock[i].get_field("zip") «" to "« index «" column.\n";}
00400     stateZips[index].push_back(recordBlock[i].get_field("zip"));
00401     //if(DEBUG){cout « stateZips[index].at(stateZips[index].size())«" pushed successfully.\n";}
00402
```

```
00403
00404     if(DEBUG){cout « stateZips[index].at(0) « ": "
00405         « stateZips[index].at(stateZips[index].size()-1) « endl;}
00406       }
00407     }
00408   }
00409 }// End addBlockStateKey
00410
00411
00412 bool SequenceSet::deleteRecord(int pKey)
00413 {
00414   //search if the record is in the sequence set
00415   int position = binarySearchSS( to_string(pKey) );
00416   if(DEBUG) {cout « "Searching for "« pKey « " returned: " « position « endl;}
00417   if(position == -1){
00418     cout « "Record does not exist in Sequence Set." « endl;
00419     return false;
00420   }
00421   else{
00422     //add deleted record offset to avail list
00423     string strTemp = "";
00424     string newString = "";
00425     fstream recordAvailListIn;
00426     recordAvailListIn.open(recordAvailListFileName);
00427     while(recordAvailListIn.peek() != EOF){
00428         strTemp += recordAvailListIn.get();
00429         if(DEBUG){cout « strTemp « endl;}
00430     }
00431     newString = to_string( offsetIndex.at(position) ) + "/" + to_string( position ) + "\n" + strTemp;
00432     if(DEBUG){cout « newString « " result" « endl;}
00433     recordAvailListIn.close();
00434
00435     ofstream recordAvailList;
00436     recordAvailList.open(recordAvailListFileName);
00437     recordAvailList « newString;
00438     recordAvailList.close();
00439
00440     //delete record from us_postal_codes.txt
00441     fstream usPostalCodes;
00442     usPostalCodes.open("us_postal_codes.txt");
00443     usPostalCodes.seekg(offsetIndex.at(position));
00444     for(int i = 0; i < 94; i++){ //94 is the length of record
00445       usPostalCodes « " ";
00446     }
00447     usPostalCodes.close();
00448
00449     //delete record in index vector
00450     pKeyIndex.erase(pKeyIndex.begin() + position);
00451     offsetIndex.erase(offsetIndex.begin() + position);
00452       if(DEBUG) {position = binarySearchSS( to_string(pKey) );}
00453     if(DEBUG) {cout « "Deleted record in index vector. Researching for "« pKey « " returned: " «
      position « endl;}
00454     recordCount--; //decrement the total record count
00455
00456     //delete record in linked list of blocks
00457     Block * currentBlock = headBlock;
00458     for(auto i = 0; i < blockCount; i ++){
00459       if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
00460       if( pKey <= currentBlock->getLastRecordPKey() ){
00461         currentBlock->deleteRecord( to_string(pKey) );
00462         break;
00463       }
00464       else{
00465         currentBlock = currentBlock->getNextBlock();
00466       }
00467     }
00468
00469     //merge blocks if needed
00470     if( currentBlock->getRecordCount() < RECORDSPERBLOCK / 2 ){
00471       //check next block to see if it can merge
00472       if( (currentBlock->getNextBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00473         currentBlock->getRecords( recordBlock ); //get the pkeys
00474         for(int i = 0; i < currentBlock->getRecordCount(); i++){
00475           (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00476           currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00477         }
00478           //add the pointer to the current block to the avail vector
00479           blockAvailList.push_back( currentBlock );
00480           //change the pointers to avoid the empty block
00481           currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00482           currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00483           blockCount--;
00484       }
00485       //check if previous block can merge
00486       else if( (currentBlock->getPreviousBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00487         currentBlock->getRecords( recordBlock ); //get the pkeys
00488         for(int i = 0; i < currentBlock->getRecordCount(); i++){
```

```
00489           (currentBlock->getPreviousBlock())->addRecord(recordBlock[i].get_field("zip"));
00490           currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00491         }
00492           //add the pointer to the current block to the avail vector
00493           blockAvailList.push_back( currentBlock );
00494           //change the pointers to avoid the empty block
00495           currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00496           currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00497           blockCount--;
00498       }
00499       //check if next block can redistribute
00500       else if( (currentBlock->getNextBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
00501         (currentBlock->getNextBlock())->getRecords( recordBlock ); //get the pkeys
00502         currentBlock->addRecord( recordBlock[0].get_field("zip") );
00503         (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
00504       }
00505       //check if previous block can redistribute ???Will Never
      Happen??????????????????????????????????????????????????????????????????????????????????????
00506       // else if( (currentBlock->getPreviousBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
00507       //   (currentBlock->getPreviousBlock())->getRecords( recordBlock ); //get the pkeys
00508       //   currentBlock->addRecord( recordBlock[0].get_field("zip") );
00509       //   (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
00510       //}
00511     }
00512     rewriteSSFile();
00513     return true;
00514   }
00515 }// End deleteRecord
00516
00517 string SequenceSet::extremeCoord(string state, char direction)
00518 {
00519   direction = toupper(direction);
00520     float extremePoint = 0;
00521     string zip = "";
00522   for(int i = 0; i < 2; i++){
00523     zip+=toupper(state[i]);
00524   }
00525   state = zip;
00526     zip = "";
00527   Record currentRecord;
00528   string str = state;
00529
00530   bool found = false;
00531   unsigned int index = 0;
00532   while(index < stateZips.size() - 1 && !found){
00533     if(stateZips[index][0] == str){found = true;}
00534     else{index++;}
00535   }
00536   currentRecord = fillRecord(fetch(stateZips[index][1]));
00537
00538     switch(direction)
00539     {
00540         case 'N':
00541         {
00542       extremePoint = stof(currentRecord.get_field("Lat"));
00543       zip = currentRecord.get_field("zip");
00544             for(int i = 1; i < stateZips[index].size(); i++)
00545             {
00546                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00547                 if(extremePoint < stof(currentRecord.get_field("Lat")))
00548                 {
00549           zip = currentRecord.get_field("zip");
00550                     extremePoint = stof(currentRecord.get_field("Lat"));
00551                 }
00552             }
00553         }
00554         break;
00555
00556         case 'E':
00557         {
00558             extremePoint = stof(currentRecord.get_field("Long"));
00559             zip = currentRecord.get_field("zip");
00560             for(int i = 1; i < stateZips[index].size(); i++)
00561             {
00562                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00563                 if(extremePoint < stof(currentRecord.get_field("Long")))
00564                 {
00565           zip = currentRecord.get_field("zip");
00566                     extremePoint = stof(currentRecord.get_field("Long"));
00567                 }
00568             }
00569         }
00570         break;
00571
00572         case 'S':
00573         {
00574             extremePoint = stof(currentRecord.get_field("Lat"));
```

```
00575             zip = currentRecord.get_field("zip");
00576             for(int i = 1; i < stateZips[index].size(); i++)
00577             {
00578                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00579                 if(extremePoint > stof(currentRecord.get_field("Lat")))
00580                 {
00581           zip = currentRecord.get_field("zip");
00582                     extremePoint = stof(currentRecord.get_field("Lat"));
00583                 }
00584             }
00585         }
00586         break;
00587 
00588         case 'W':
00589         {
00590             extremePoint = stof(currentRecord.get_field("Long"));
00591             zip = currentRecord.get_field("zip");
00592             for(int i = 1; i < stateZips[index].size(); i++)
00593             {
00594                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00595                 if(extremePoint > stof(currentRecord.get_field("Long")))
00596                 {
00597           zip = currentRecord.get_field("zip");
00598                     extremePoint = stof(currentRecord.get_field("Long"));
00599                 }
00600             }
00601         }
00602         break;
00603 
00604     default:
00605     {
00606       cout << "UNDEFINED OPTION\n";
00607     }
00608     }
00609     return zip;
00610 }// End extremeCoord
00611 
00612 int SequenceSet::test(){
00613     string field;
00614     string str = "";
00615     char c;
00616     fstream data;
00617 
00618     int randomRecord = rand() % recordCount;
00619     //cout << "Retrieving record: " << index[randomRecord][0] << endl;
00620     cout << "Retrieving record: " << pKeyIndex.at(randomRecord) << endl;
00621     data.open(DATAFILENAME);
00622     //data.seekg(index[randomRecord][1]);
00623     data.seekg(offsetIndex.at(randomRecord));
00624     getline(data, str);
00625     cout << str << endl;
00626 
00627     cout << fetch(1721) << endl;
00628     fillRecordBlock(88);
00629 
00630     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00631       if(DEBUG){cout <<"\n**********************************\n";}
00632       recordBlock[i].display();
00633     }
00634 
00635     sKeyStateBuilder();
00636 
00637     unsigned int index = 0;
00638     unsigned int record = 1;
00639     Record currentRecord;
00640 
00641     str = "MN";
00642     bool found = false;
00643     while(index < stateZips.size() && !found){
00644       if(stateZips[index][0] == str){found = true;}
00645       else{index++;}
00646     }
00647 
00648     while(record < stateZips[index].size()){
00649       str = fetch(stateZips[index][record]);
00650       cout << str << endl;
00651       currentRecord = fillRecord(str);
00652       currentRecord.display();
00653       record++;
00654     }
00655 
00656     cout << extremeCoord(str, 'n') << endl;
00657 
00658     return 0;
00659 }// End test
00660 
00661 
```

```
00662 void SequenceSet::sKeyStateBuilder(){
00663     if(DEBUG){cout « "Building sKeys for states.\n";}
00664     Block * currentBlock = headBlock;
00665     unsigned int index = 0;
00666     while(currentBlock!=NULL){
00667       addBlockStateKey(index);
00668       currentBlock = currentBlock->getNextBlock();
00669       index++;
00670     }
00671 }//End sKeyStateBuilder
00672
00673 void SequenceSet::addRecord(Record record)
00674 {
00675     //search record in linked list of blocks
00676     Block * currentBlock = headBlock;
00677     for(auto i = 0; i < blockCount; i ++){
00678         if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
00679         if( stoi( record.get_field("zip") ) <= currentBlock->getLastRecordPKey() ){ //find the right
      block
00680       if(currentBlock->getRecordCount() == RECORDSPERBLOCK){ //if the block is full, do block
      splitting
00681         if( !blockAvailList.empty() ){  //if there exists a current empty block
00682           Block* tempBlockPtr = blockAvailList.back(); //get the pointer to the empty block
00683           blockAvailList.pop_back();  //delete the pointer from the avail list
00684           //add the relative block to the linked list
00685           tempBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00686           tempBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
00687           (currentBlock->getNextBlock())->setPrevBlock(tempBlockPtr);
00688           currentBlock->setNextBlock(tempBlockPtr);
00689           //split the data into the new block number
00690           currentBlock->getRecords( recordBlock ); //get the pkeys
00691           for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00692             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00693             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00694           }
00695           //add the new record to the block
00696           currentBlock->addRecord( record.get_field("zip") );
00697           blockCount++;
00698           break; //stop searching through linked list of blocks
00699         }
00700         else{  //if a current empty block doesn't exist, create a new block......
00701           Block* newBlockPtr = new Block;
00702           newBlockPtr->setRBN(blockCount);
00703           newBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00704           newBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
00705           (currentBlock->getNextBlock())->setPrevBlock(newBlockPtr);
00706           currentBlock->setNextBlock(newBlockPtr);
00707           //split the data into the new block number
00708           currentBlock->getRecords( recordBlock ); //get the pkeys
00709           for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00710             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00711             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00712           }
00713           //add the new record to the block
00714           currentBlock->addRecord( record.get_field("zip") );
00715           blockCount++;
00716           break; //stop searching through linked list of blocks
00717         }
00718       }
00719       else{
00720         currentBlock->addRecord( record.get_field("zip") );
00721       }
00722             break; //stop searching through linked list of blocks
00723         }
00724         else{
00725             currentBlock = currentBlock->getNextBlock();
00726         }
00727     }
00728
00729   //add record to us_postal_codes.txt
00730   fstream recordAvailList;
00731   string str = "";
00732   string strTemp = "";
00733   string offset = "";
00734   string position = "";
00735   recordAvailList.open(recordAvailListFileName);
00736   if( recordAvailList.peek() != EOF ){ //if recordAvailList is not empty
00737     fstream usPostalCodes;
00738     usPostalCodes.open("us_postal_codes.txt");
00739     getline(recordAvailList, str); //get the offset and vector position from avail list
00740     int i = 0;
00741     while( str[i] != '/' ){ //parse the offset from the string
00742         offset += str[i];
00743         if(DEBUG){cout « offset « endl;}
00744         i++;
00745     }
00746     i++;
```

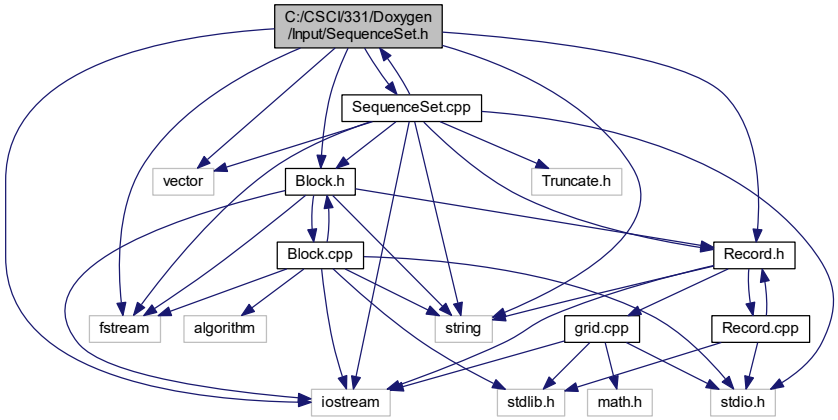
```
00747     while( i < str.length()  ){ //parse the position from the string
00748         position += str[i];
00749         if(DEBUG){cout « position « endl;}
00750         i++;
00751     }
00752     writeToTxt(record, offset, "us_postal_codes.txt");
00753     usPostalCodes.close();
00754     recordAvailList.close();
00755     //update the recordAvailList
00756     recordAvailList.open(recordAvailListFileName);
00757     str += "\n";
00758     if(DEBUG){cout « str « " str to delete" « endl;}
00759     while(recordAvailList.peek() != EOF){
00760         strTemp += recordAvailList.get();
00761         if(DEBUG){cout « strTemp « endl;}
00762         if(strTemp == str){
00763             strTemp = "";
00764         }
00765     }
00766     recordAvailList.close();
00767     remove("availRecordList.txt");
00768     ofstream recordAvailListOut;
00769     recordAvailListOut.open(recordAvailListFileName, ios::app);
00770     if(DEBUG){cout « strTemp « " result" « endl;}
00771     recordAvailListOut « strTemp;
00772     recordAvailListOut.close();
00773     //add record to index vector
00774     if(DEBUG){for(int i=0; i<20; ++i)std::cout « pKeyIndex[i] « ' ';}
00775     pKeyIndex.insert(pKeyIndex.begin() + stoi( position ), stoi( record.get_field("zip") ) );
00776     offsetIndex.insert(offsetIndex.begin() + stoi( position ), stoi( offset ) );
00777     cout «endl;
00778     if(DEBUG){for(int i=0; i<20; ++i)std::cout « pKeyIndex[i] « ' ';}
00779   }
00780   else{ //if recordAvailList is empty
00781     unsigned int nextOffset = offsetIndex.back() + 95;//95 is record length+1
00782     if(DEBUG){cout « nextOffset « " nextoffset" « endl;}
00783     pKeyIndex.push_back( stoi( record.get_field("zip") ) );
00784     offsetIndex.push_back( nextOffset );
00785     writeToTxt(record, to_string( nextOffset ), "us_postal_codes.txt");
00786     ofstream usPostalCodes;
00787     usPostalCodes.open("us_postal_codes.txt", ios::app);
00788     usPostalCodes « endl;
00789     usPostalCodes.close();
00790   }
00791
00792   rewriteSSFile();
00793 }// End addRecord
00794
00795 void SequenceSet::rewriteSSFile()
00796 {
00797     //rewrite the squence set file with missing record
00798     remove("Sequence_Set.txt");
00799     ofstream SSFile;
00800     SSFile.open(SSFileName);
00801     SSFile « "Sequence Set File\n";
00802     SSFile.close();
00803     writeBlocks();
00804 }//End rewriteSSFile
00805
00806 //write the record to the postal codes file
00807 void SequenceSet::writeToTxt(Record record, string offset, string _fileName)
00808 {
00809     fstream data;
00810     data.open(_fileName);
00811     data.seekg( stoi( offset ) );
00812
00813     string dataString = "";
00814     string totalString = "";
00815
00816     dataString = record.get_field("Zip");
00817     int fieldLength = 6;
00818     for(int i = 0; i < fieldLength - dataString.length(); i++){
00819         totalString += " ";
00820     }
00821     totalString += dataString;
00822
00823     dataString = record.get_field("city");
00824     fieldLength = 31;
00825     totalString += dataString;
00826     for(int i = 0; i < fieldLength - dataString.length(); i++){
00827         totalString += " ";
00828     }
00829
00830     dataString = record.get_field("state");
00831     totalString += dataString;
00832
00833     dataString = record.get_field("county");
```

```
00834     fieldLength = 38;
00835     totalString += dataString;
00836     for(int i = 0; i < fieldLength - dataString.length(); i++){
00837         totalString += " ";
00838     }
00839
00840     dataString = record.get_field("long");
00841     while(dataString.length() > 7){
00842         dataString.pop_back();
00843     }
00844     fieldLength = 8;
00845     for(int i = 0; i < fieldLength - dataString.length(); i++){
00846         totalString += " ";
00847     }
00848     totalString += dataString;
00849
00850     dataString = record.get_field("lat");
00851     fieldLength = 9;
00852     while(dataString.length() > 8){
00853         dataString.pop_back();
00854     }
00855     for(int i = 0; i < fieldLength - dataString.length(); i++){
00856         totalString += " ";
00857     }
00858     totalString += dataString;
00859
00860     data « totalString;
00861
00862     data.close();
00863 }// End writeToTxt
```

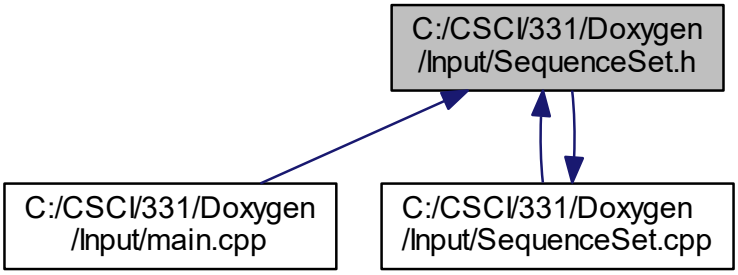
4.17 C:/CSCI/331/Doxygen/Input/SequenceSet.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "Block.h"
#include "Record.h"
#include "SequenceSet.cpp"
```

Include dependency graph for SequenceSet.h:



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



Classes

- class SequenceSet

4.18 SequenceSet.h

```
00001
00019 #ifndef SEQUENCESET_H
00020 #define SEQUENCESET_H
00021
00022 #include <iostream>
00023 #include <string>
00024 #include <fstream>
00025 #include <vector>
00026
00027 #include "Block.h"
00028 #include "Record.h"
00029
00030 using namespace std;
00031
00032 class SequenceSet
00033 {
00034 private:
00035     string SSFileName = "Sequence_Set.txt";
00036     string recordAvailListFileName = "availRecordList.txt";
00037     unsigned long long headerLength(string);
00038     unsigned long long blockCount;
00039     unsigned int recordCount;
00040     //unsigned int indexArray[getRecordCount()][2];
00041     vector<unsigned int>pKeyIndex;
00042     vector<unsigned int>offsetIndex;
00046     vector<vector<string»stateZips;
00047     vector<vector<string»sKeyCounty;
00048     vector<vector<string»sKeyPlace;
00049     vector<Block*>blockAvailList;
00050     Record recordBlock[RECORDSPERBLOCK];
00051     Block * headBlock = new Block;
00057     void addBlockStateKey(unsigned long long blockID);
00058
00063     void sKeyStateBuilder();
00064
00069     int binarySearchSS(string x);
00070
00071 public:
00072
00073     DATAFILENAME and HEADERENDSTRING
00074
00079     SequenceSet();
00080
00086     void makeRecordOffsets(string fileName);
00087
00093     void fillIndex();
```

```
00094
00099     void fillRecordBlock(unsigned long long blockID);
00100
00105     void writeBlocks();
00106
00112     Record fillRecord(string RecordString);
00113
00119     unsigned int getRecordCount();
00120
00125     string fetch(string pKey);
00126
00131     string fetch(unsigned int pKey);
00132
00138     string extremeCoord(string, char);
00139
00144     bool deleteRecord(int pKey);
00145
00150     void addRecord(Record record);
00151     void rewriteSSFile();
00152     void writeToTxt(Record, string, string);
00153 };
00154
00155 #include "SequenceSet.cpp"
00156
00157 #endif
00158
```

Index