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>`

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 addRecord()

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

Precondition

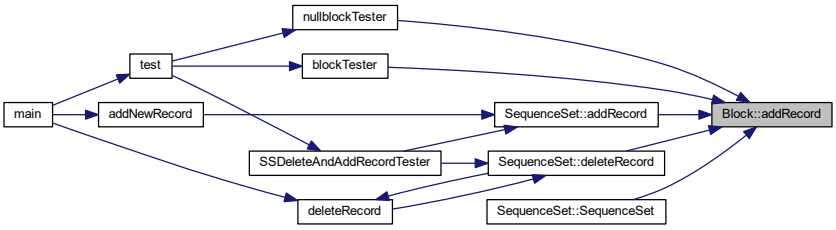
Primary key

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:



3.1.3.2 blockData()

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

Returns RBN and records of the block.

Definition at line 70 of file Block.cpp.

3.1.3.3 deleteRecord()

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

Precondition

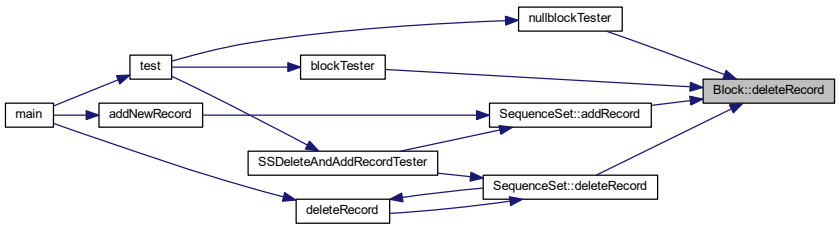
Primary key

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:



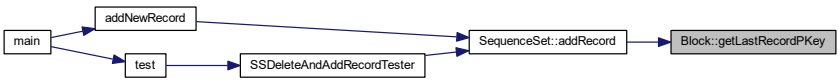
3.1.3.4 getLastRecordPKey()

```
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:



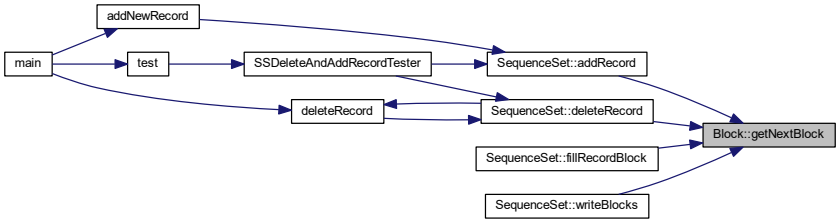
3.1.3.5 getNextBlock()

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

Gets pointer of next block.

Definition at line 192 of file Block.cpp.

Here is the caller graph for this function:



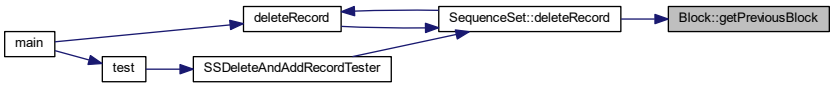
3.1.3.6 getPreviousBlock()

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

Gets pointer of previous block.

Definition at line 199 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.7 getRBN()

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

Gets the relative block number.

Definition at line 302 of file Block.cpp.

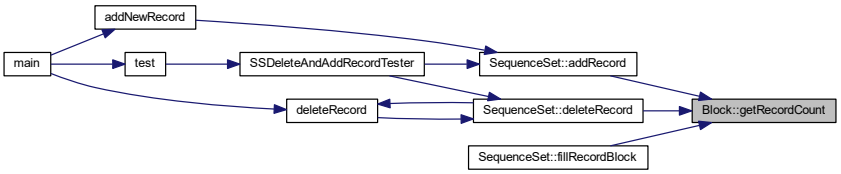
3.1.3.8 getRecordCount()

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

Gets the record count.

Definition at line 220 of file Block.cpp.

Here is the caller graph for this function:



3.1.3.9 getRecords()

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

Precondition

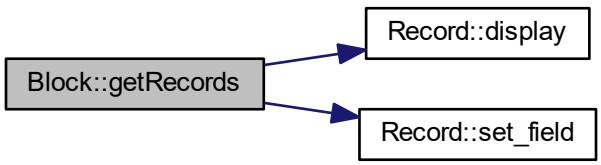
Record object array

Postcondition

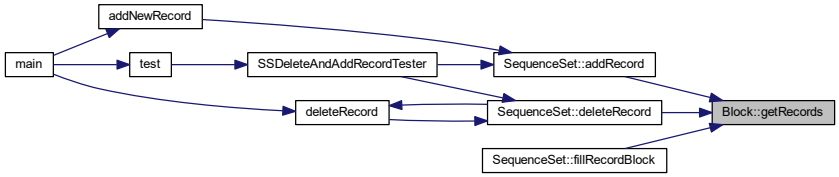
Fills record block

Definition at line 289 of file Block.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.10 search()

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

Searches for record.

Precondition

Primary key

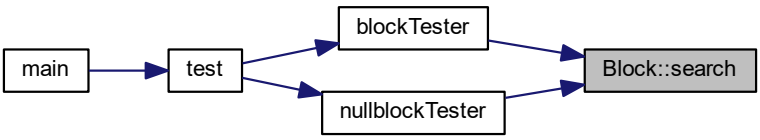
Postcondition

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:



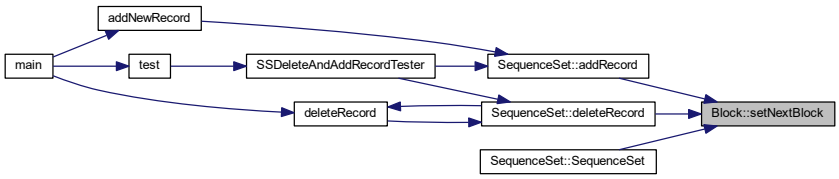
3.1.3.11 setNextBlock()

```
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:



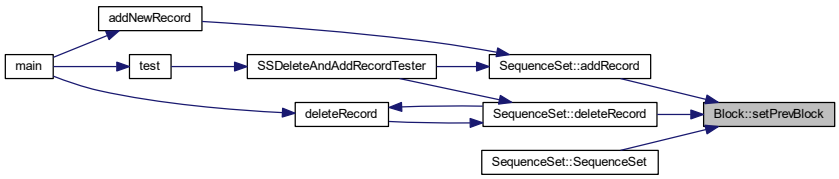
3.1.3.12 setPrevBlock()

```
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:



3.1.3.13 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:

3.1.3.14 write()

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

Precondition

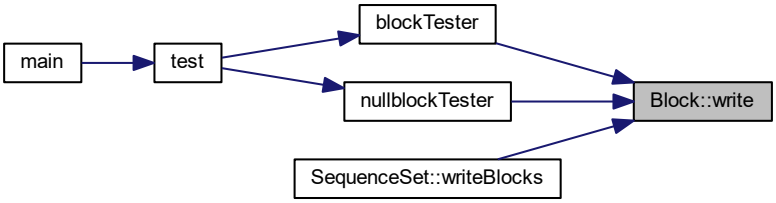
A block

Postcondition

Writes the block to a file

Definition at line 146 of file Block.cpp.

Here is the caller graph for this function:



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

- Doxygen/Input/Block.h
- Doxygen/Input/Block.cpp

3.2 Grid Class Reference

Grid class.

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 getDistance()

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

Gets Distance from this grid object to another grid object.

Precondition

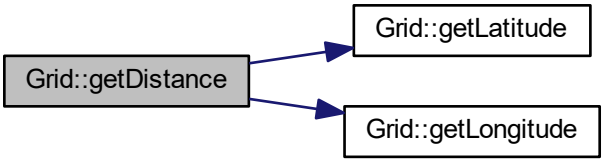
grid object must be provided

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:



3.2.3.2 getLatitude()

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

Gets Latitude for this grid object.

Precondition

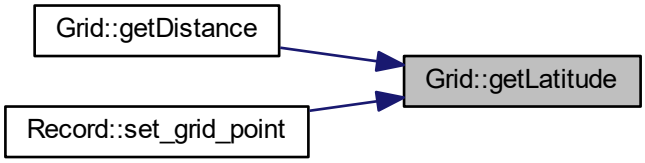
none

Postcondition

returns latitude for grid object as float

Definition at line 101 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.3 getLongitude()

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

Gets Longitude for this grid object.

Precondition

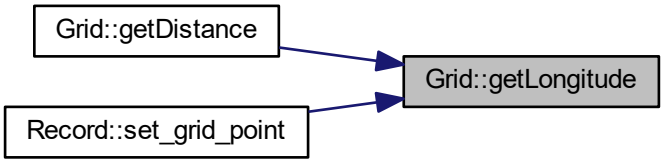
none

Postcondition

returns longitude for grid object as float

Definition at line 109 of file grid.cpp.

Here is the caller graph for this function:



3.2.3.4 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.

3.2.3.5 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.

3.2.3.6 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.

3.2.3.7 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.

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

- Doxygen/Input/grid.cpp

3.3 Record Class Reference

```
#include <Record.h>
```

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

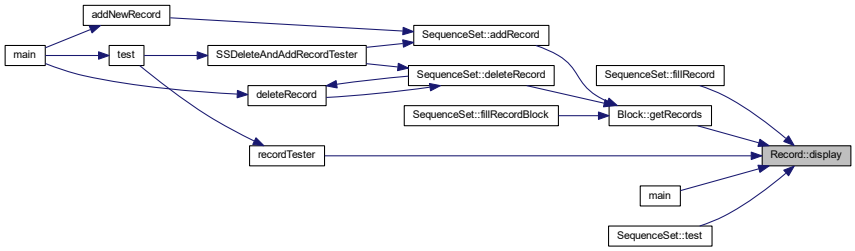
None

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

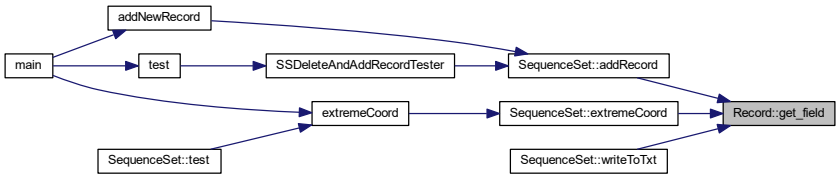
Provided string must match the name of a field in the record

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

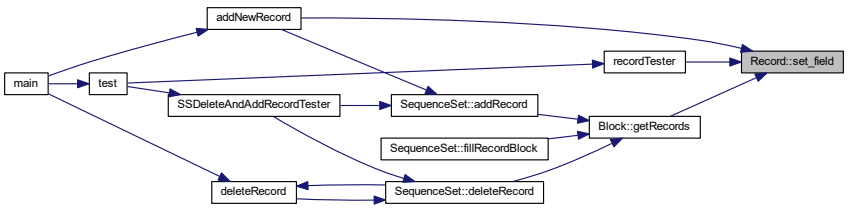
First provided string must match the name of a field in the record Second provided string must be the appropriate length for the field

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_grid_point()

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

Sets the Latitude and longitude based on a grid point.

Precondition

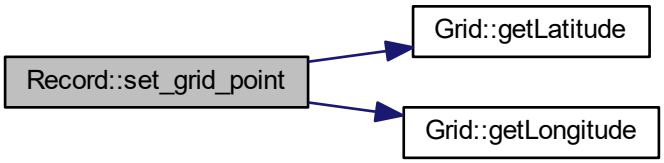
A grid point of type Grid

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:



3.3.3.6 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.

Here is the caller graph for this function:

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

- Doxygen/Input/Record.h
- Doxygen/Input/Record.cpp

3.4 SequenceSet Class Reference

`#include <SequenceSet.h>`

Public Member Functions

- 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)

  WriteToTxt.
- int binarySearchSS (string x)

  Searches block for record by primary key.
- int test ()

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

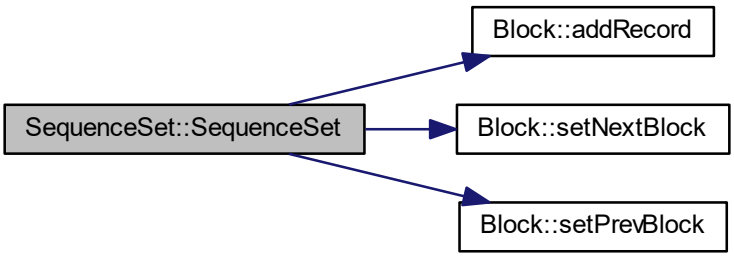
File with a name that matches DATAFILENAME in Header.cpp must exist and it must contain a line equal to HEADERENDSTRING in Header.cpp

Postcondition

Sequence Set Class is made

Definition at line 35 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3 Member Function Documentation

3.4.3.1 addRecord()

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

Add record.

Precondition

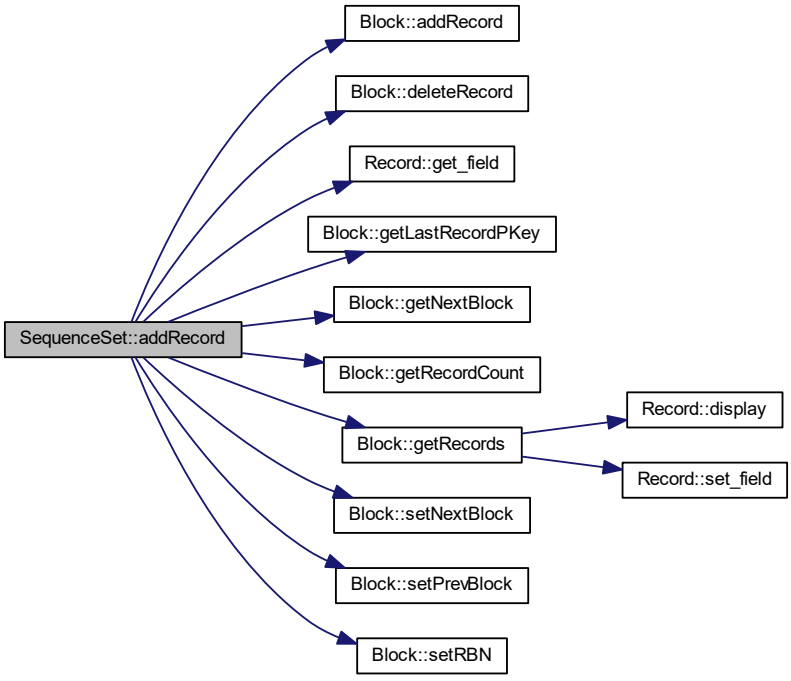
A record object

Postcondition

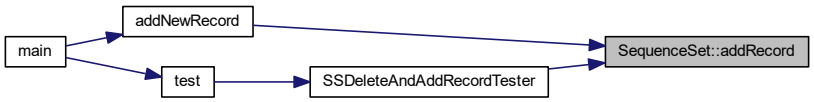
Adds the record

Definition at line 672 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.2 binarySearchSS()

```
int SequenceSet::binarySearchSS (
            string x )
```

Searches block for record by primary key.

Precondition

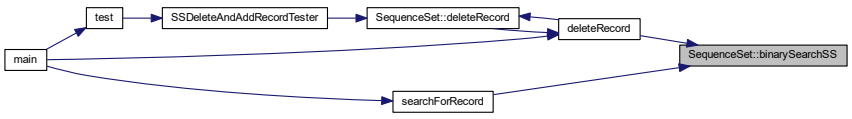
Primary key

Postcondition

Returns true if found otherwise returns false

Definition at line 215 of file SequenceSet.cpp.

Here is the caller graph for this function:



3.4.3.3 deleteRecord()

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

Delete record.

Precondition

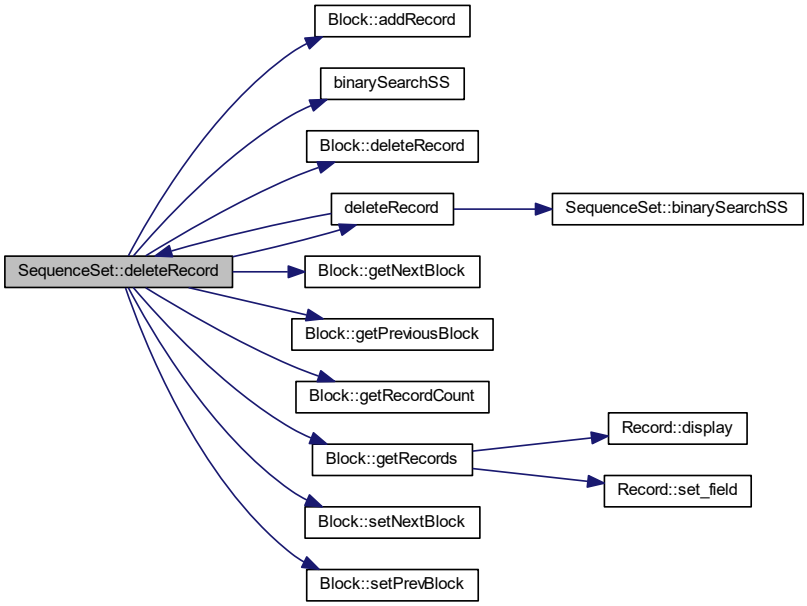
A primary key (zipcode)

Postcondition

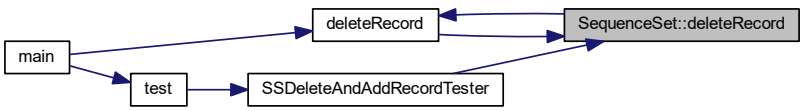
Deletes record with given zipcode

Definition at line 417 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.4 extremeCoord()

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

Extreme coordinate.

Precondition

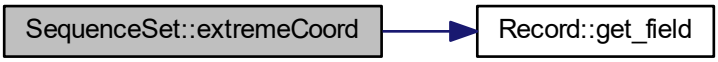
State of type string and Direction of type Char (N, E, S, W) State code must be in the list of states or the last state in list is used

Postcondition

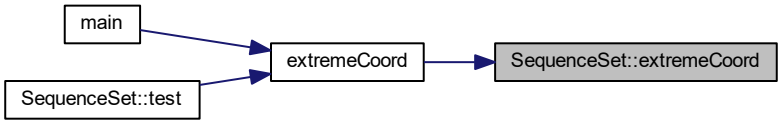
Returns the zipcode containing the most extreme point of said direction

Definition at line 516 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.5 fetch() [1/2]

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

Fetch string.

Precondition

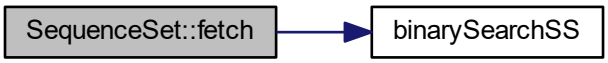
None

Postcondition

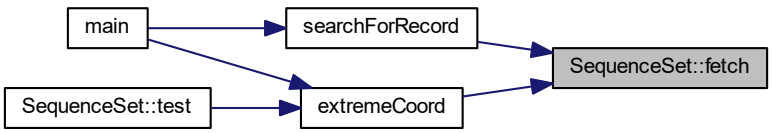
returns the whole record as a string

Definition at line 156 of file SequenceSet.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.6 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 181 of file SequenceSet.cpp.

3.4.3.7 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 124 of file SequenceSet.cpp.

3.4.3.8 fillRecord()

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

Fill record.

Precondition

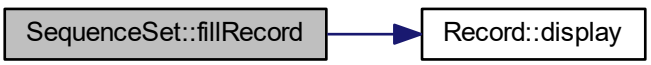
Record string must follow parameter conventions Record string must be complete, call fetch if needed

Postcondition

A record string is loaded into a record object

Definition at line 264 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.9 fillRecordBlock()

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

Fill record block.

Precondition

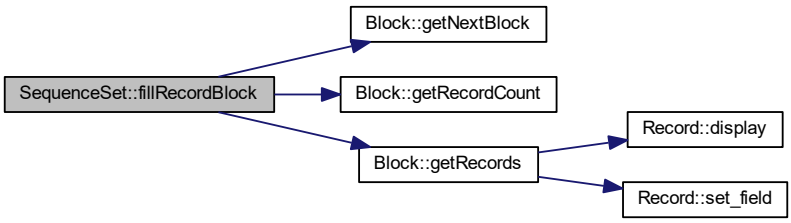
blockID must be less than the block count

Postcondition

Block is loaded into a record block

Definition at line 337 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.10 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 90 of file SequenceSet.cpp.

3.4.3.11 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

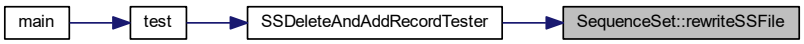
Definition at line 185 of file SequenceSet.cpp.

3.4.3.12 rewriteSSFile()

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

Definition at line 794 of file SequenceSet.cpp.

Here is the caller graph for this function:



3.4.3.13 test()

```
int SequenceSet::test ( )
```

Precondition

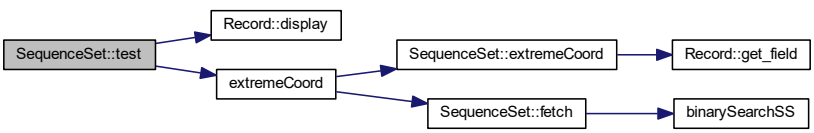
None

Postcondition

Returns the various tester results

Definition at line 611 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.14 writeBlocks()

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

Write blocks.

Precondition

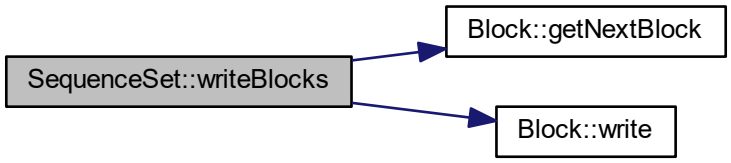
None

Postcondition

All blocks are called to run their write function

Definition at line 328 of file SequenceSet.cpp.

Here is the call graph for this function:



3.4.3.15 writeToTxt()

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

WriteToTxt.

Precondition

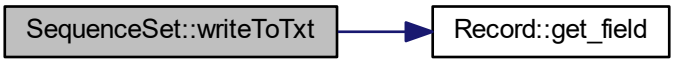
Inputs should match source length requirements

Postcondition

Writes the record to the data file (us_postal_codes.txt)

Definition at line 806 of file SequenceSet.cpp.

Here is the call graph for this function:



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

- Doxygen/Input/SequenceSet.h
- Doxygen/Input/SequenceSet.cpp

3.5 Truncate Class Reference

`#include <Truncate.h>`

Public Member Functions

- Truncate ()

  Default constructor Preconditions: None Postconditions: A truncate object will be created with a size of default for max length.
- Truncate (int)

  Default constructor Preconditions: Input must be an int Postconditions: A truncate object will be created with a size of the input for max length.
- string modifyString (string &)

  String modifier Preconditions: Input must be a string Postconditions: The truncate object will truncate the input string and return it.
- string truncatedString (string)

  Temporary string modifier Preconditions: Input must be a string Postconditions: The truncate object will copy the input string and return the truncated string without modifying the original.

3.5.1 Detailed Description

Definition at line 7 of file Truncate.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Truncate() [1/2]

```
Truncate::Truncate ( )
```

Default constructor Preconditions: None Postconditions: A truncate object will be created with a size of default for max length.

Definition at line 10 of file Truncate.cpp.

3.5.2.2 Truncate() [2/2]

```
Truncate::Truncate (
            int _size )
```

Default constructor Preconditions: Input must be an int Postconditions: A truncate object will be created with a size of the input for max length.

Definition at line 16 of file Truncate.cpp.

3.5.3 Member Function Documentation

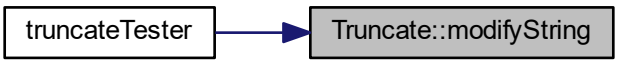
3.5.3.1 modifyString()

```
string Truncate::modifyString (
            string & _originalStr )
```

String modifier Preconditions: Input must be a string Postconditions: The truncate object will truncate the input string and return it.

Definition at line 27 of file Truncate.cpp.

Here is the caller graph for this function:



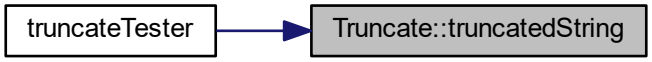
3.5.3.2 truncatedString()

```
string Truncate::truncatedString (
            string _string )
```

Temporary string modifier Preconditions: Input must be a string Postconditions: The truncate object will copy the input string and return the truncated string without modifying the original.

Definition at line 20 of file Truncate.cpp.

Here is the caller graph for this function:



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

- Doxygen/Input/Truncate.h
- Doxygen/Input/Truncate.cpp

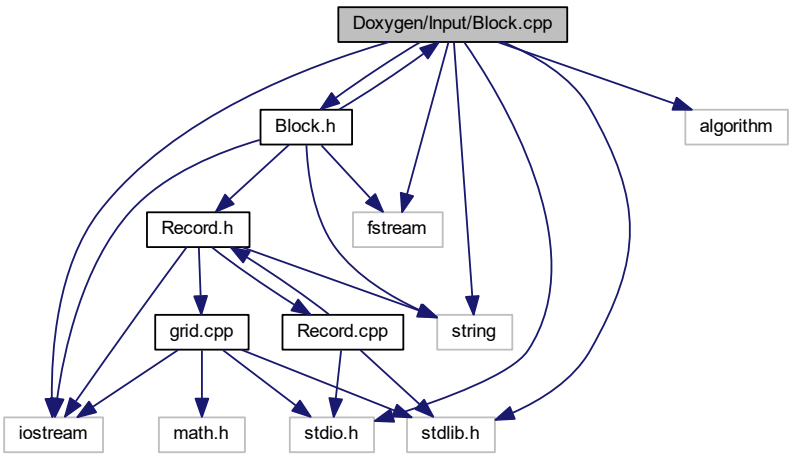
Chapter 4

File Documentation

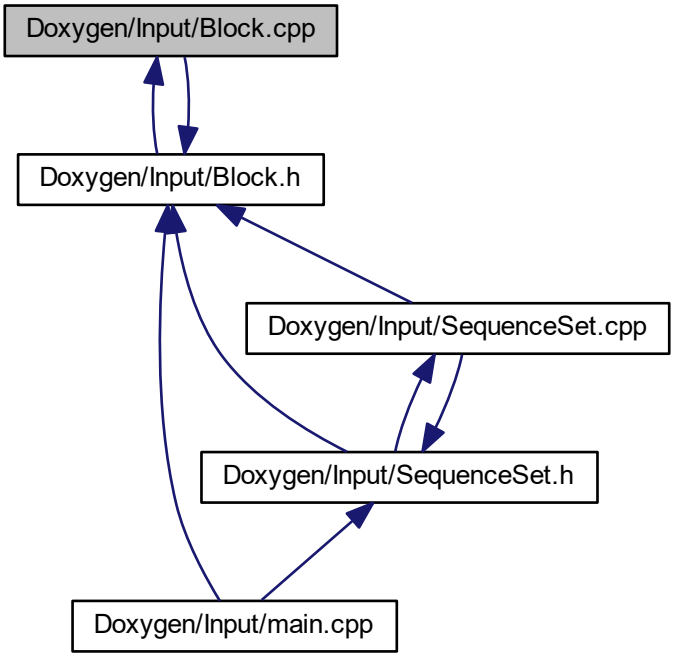
4.1 Doxygen/Input/Block.cpp File Reference

```
#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

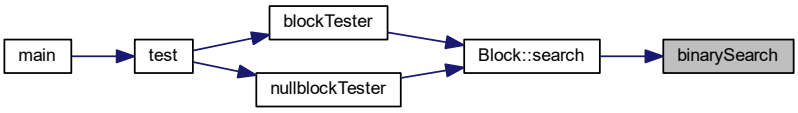
Primary key

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 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.

4.1.1.3 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.

4.1.2 Variable Documentation

4.1.2.1 NULL_INT

```
const int NULL_INT = 1000000
```

Definition at line 38 of file Block.cpp.

4.1.2.2 null_str

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

Definition at line 37 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
```

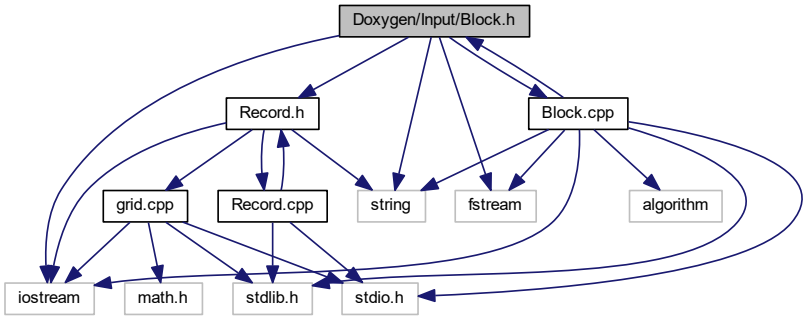
```
00277       if(DEBUG) {cout « "Elements of Constructed block after sorting record: \"" ;
00278             for(int i = 0; i < RECORDSPERBLOCK; i++){cout « records[i];}
00279             cout « "\".\n";}
00280       return true;
00281     }
00282   }
00283
00284   if(DEBUG) {cout « "Block Full. Could 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 }
```

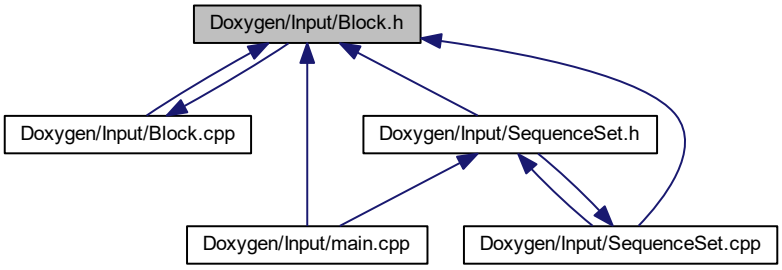
4.3 Doxygen/Input/Block.h File Reference

```
#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

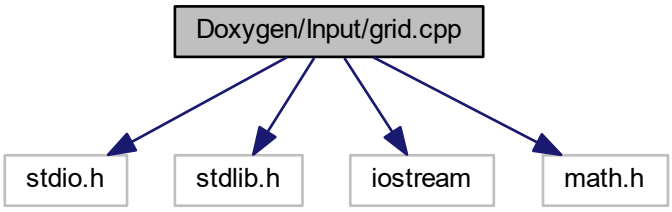
```
00001
00021 #ifndef BLOCK_H
00022 #define BLOCK_H
00023
00024 #include <iostream>
00025 #include <string>
00026 #include <fstream>
00027 #include "Record.h"
00028 using namespace std;
00029
00030 class Block
00031 {
00032   public:
00037     Block();
00038
00043     Block(unsigned long long _RBN);
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
```

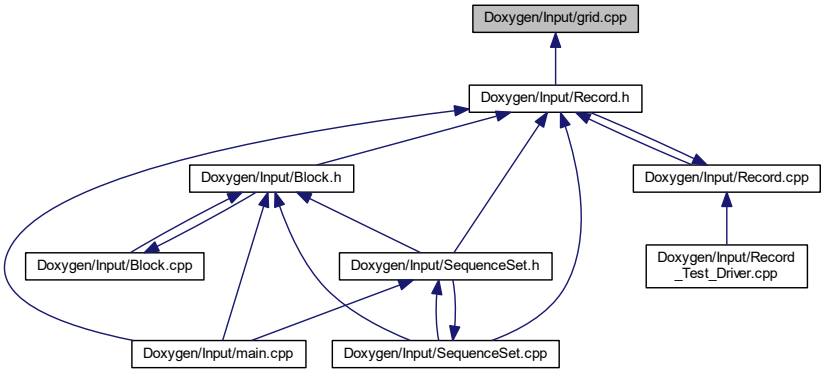
4.5 Doxygen/Input/grid.cpp File Reference

```
#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

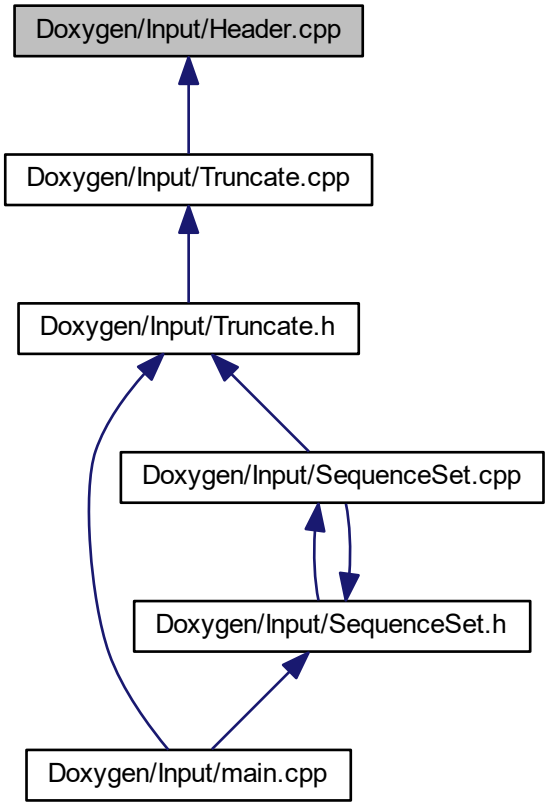
  Grid class.

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 Doxygen/Input/Header.cpp File Reference

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



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 = "12345678901234567890123456789012345678901234567890123456789012345678901234567890
  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 BLOCKFILLCOUNT

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

Max length of the RBN code in digits.

4.7.1.2 BLOCKLENGTH

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

Maximum length for the block

4.7.1.3 DATAFILENAME

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

Data file name.

4.7.1.4 DEBUG

```
const bool DEBUG = false
```

Set true for debugging output

4.7.1.5 FILLPERCENT

```
const double FILLPERCENT = 75
```

Max length of the RBN code in digits.

4.7.1.6 HEADERENDSTRING

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

String at the end of the header.

4.7.1.7 RBNLENGTH

```
const int RBNLENGTH = 8
```

Max length of the RBN code in digits.

4.7.1.8 RECORDSPERBLOCK

```
const int RECORDSPERBLOCK = 4
```

Maximum records for the block

4.7.1.9 ZIPLENGTH

```
const int ZIPLENGTH = 6
```

Max length of the zip code in digits.

4.8 Header.cpp

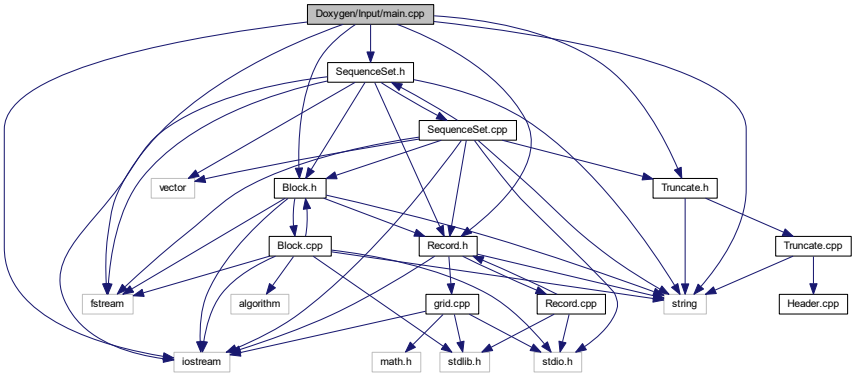
```
00001 extern const bool DEBUG               = false;
00002 extern const int RECORDSPERBLOCK = 4;
00003 extern const int ZIPLENGTH          = 6;
00004 extern const int RBNLENGTH         = 8;
00005 extern const int BLOCKLENGTH         = RBNLENGTH + RECORDSPERBLOCK * ZIPLENGTH;
00006 extern const double FILLPERCENT      = 75;
00007 extern const int BLOCKFILLCOUNT       = RECORDSPERBLOCK * (FILLPERCENT/100);
00008 extern const string HEADERENDSTRING =
       "12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890";

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

4.9 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 ()
- void test ()
- int main ()

Variables

- SequenceSet SSClass
- bool quit = false

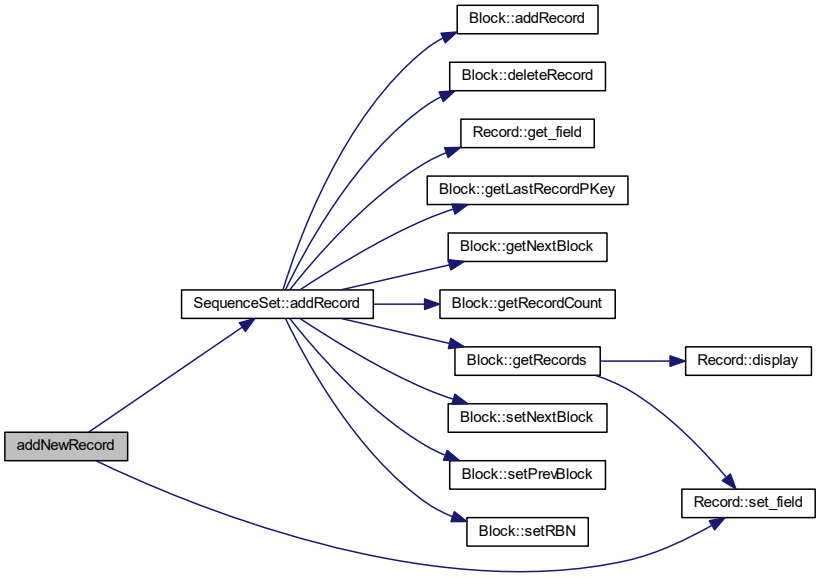
4.9.1 Function Documentation

4.9.1.1 addNewRecord()

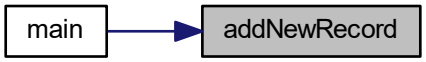
```
void addNewRecord ( )
```

Definition at line 173 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

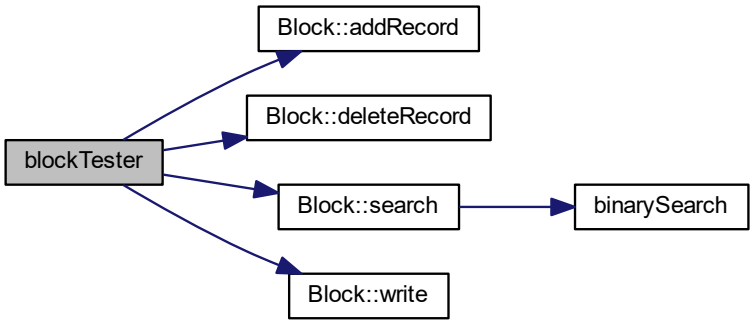


4.9.1.2 blockTester()

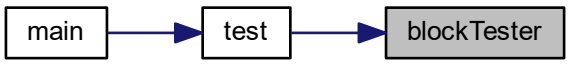
```
void blockTester ( )
```

Definition at line 378 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

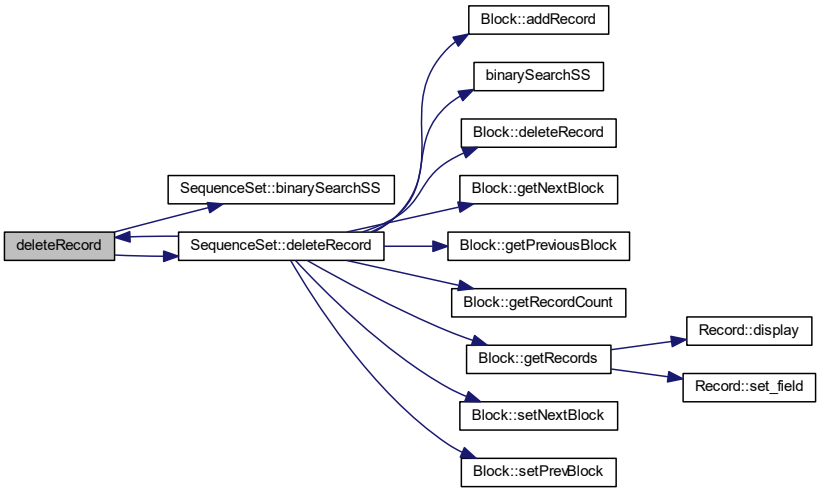


4.9.1.3 deleteRecord()

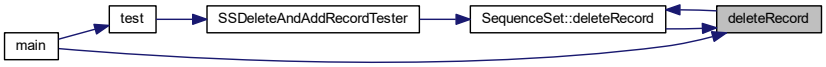
```
void deleteRecord ( )
```

Definition at line 100 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

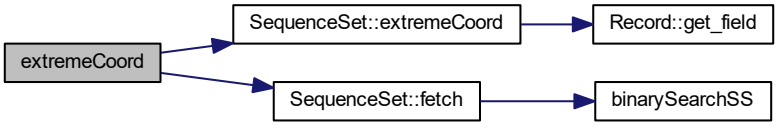


4.9.1.4 extremeCoord()

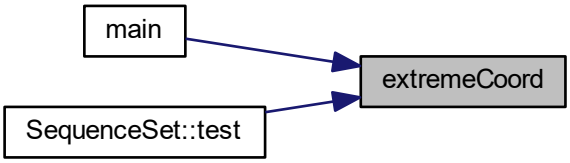
```
void extremeCoord ( )
```

Definition at line 66 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

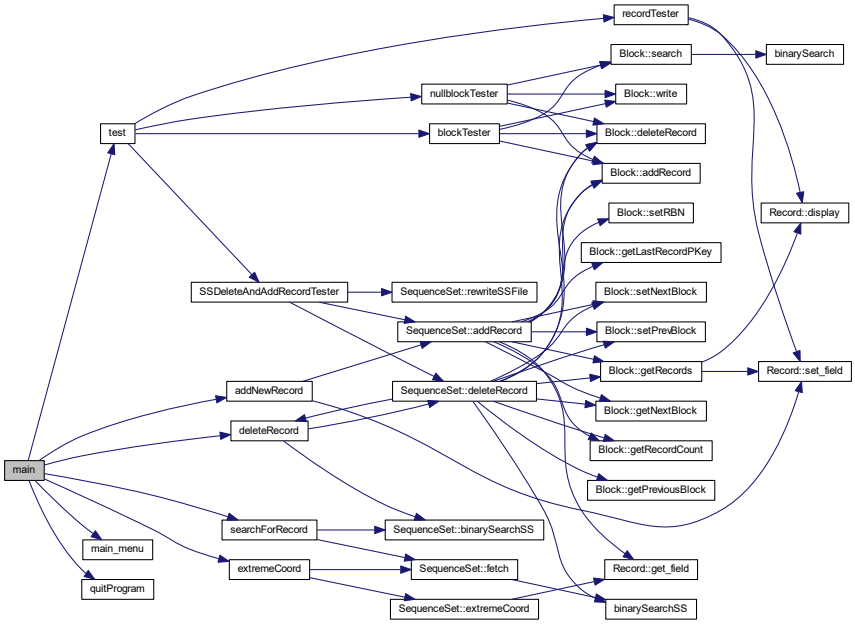


4.9.1.5 main()

```
int main ( )
```

Definition at line 27 of file main.cpp.

Here is the call graph for this function:

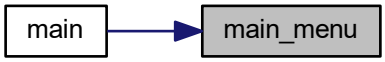


4.9.1.6 main_menu()

```
int main_menu ( )
```

Definition at line 284 of file main.cpp.

Here is the caller graph for this function:

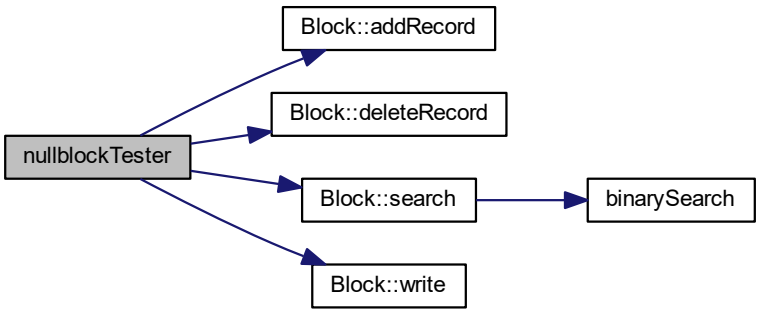


4.9.1.7 nullblockTester()

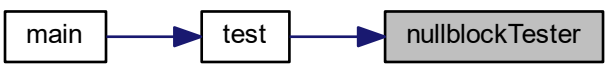
```
void nullblockTester ( )
```

Definition at line 345 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

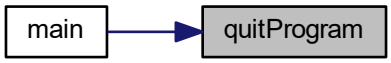


4.9.1.8 quitProgram()

```
void quitProgram ( )
```

Definition at line 95 of file main.cpp.

Here is the caller graph for this function:

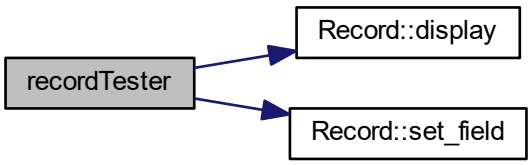


4.9.1.9 recordTester()

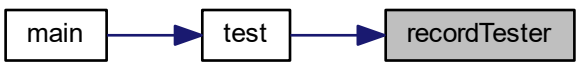
```
void recordTester ( )
```

Definition at line 421 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

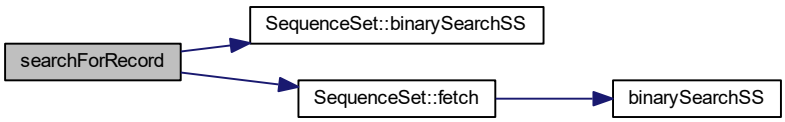


4.9.1.10 searchForRecord()

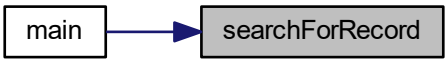
```
void searchForRecord ( )
```

Definition at line 139 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

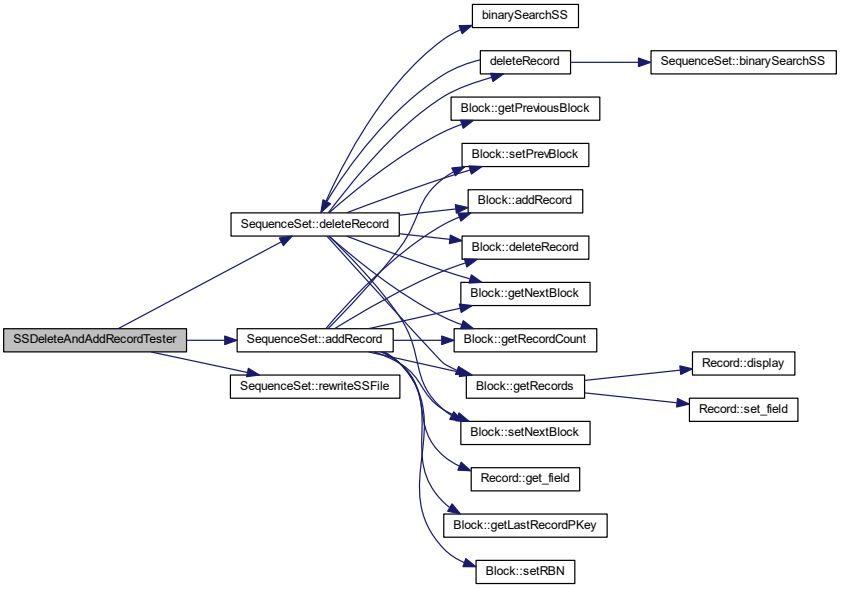


4.9.1.11 SSDeleteAndAddRecordTester()

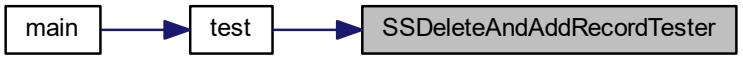
```
void SSDeleteAndAddRecordTester ( )
```

Definition at line 305 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

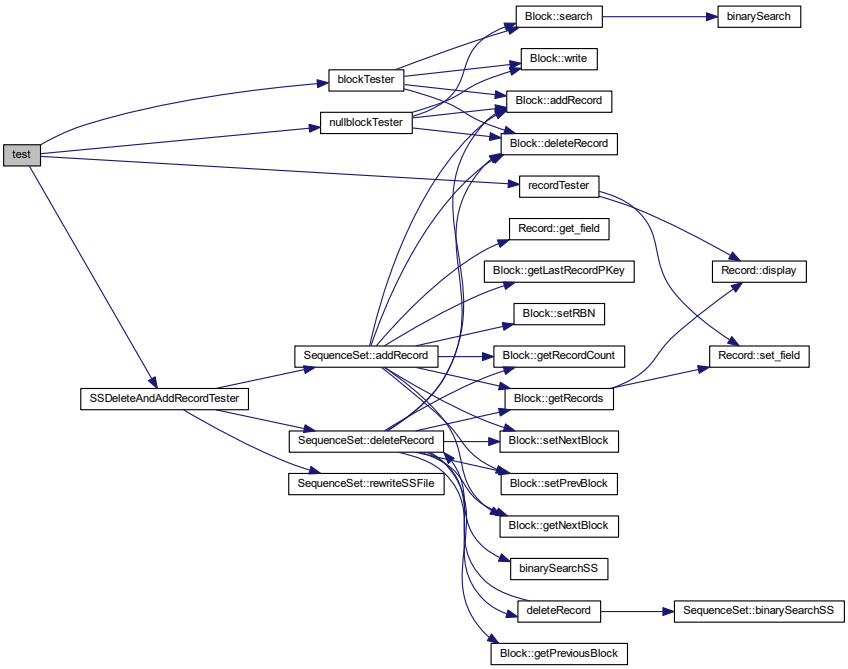


4.9.1.12 test()

```
void test ( )
```

Definition at line 59 of file main.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

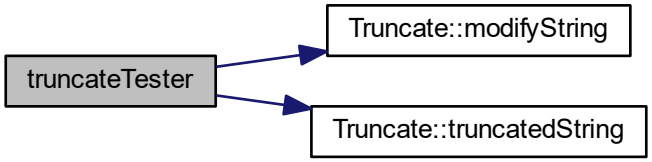
4.9.1.13 truncateTester()

```
void truncateTester ( )
```

Tests the Truncate Class.

Definition at line 410 of file main.cpp.

Here is the call graph for this function:



4.9.2 Variable Documentation

4.9.2.1 quit

```
bool quit = false
```

Definition at line 25 of file main.cpp.

4.9.2.2 SSClass

```
SequenceSet SSClass
```

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 void test();
00023
00024 SequenceSet SSClass;
```

```
00025 bool quit = false;
00026
00027 int main(){
00028
00029   int choice;
00030   cout « "Sequence Set Created." « endl;
00031
00032   while( !quit )
00033   {
00034     cout « endl « endl;
00035
00036     choice = main_menu();
00037
00038     switch( choice )
00039     {
00040       case 1: addNewRecord();
00041           break;
00042       case 2: searchForRecord();
00043           break;
00044       case 3: deleteRecord();
00045           break;
00046       case 4: quitProgram();
00047           break;
00048       case 5: extremeCoord();
00049           break;
00050       case 6: test();
00051           break;
00052       default: cout « "Selecting menu option canceled." « endl;
00053     }
00054   }
00055
00056   return 0;
00057 }
00058
00059 void test(){
00060   recordTester();
00061   blockTester();
00062   nullblockTester();
00063   SSDeleteAndAddRecordTester();
00064 }
00065
00066 void extremeCoord()
00067 {
00068   string state, dir;
00069   char direction;
00070   cout « "Please enter the state you wish to search in its two letter code with CAPS.\n\t";
00071   cin » state;
00072   cout  « "Please enter the direction you wish to find the extreme with a "
00073         «"character by choosing n, s, e, or w.\n\t";
00074   cin » direction;
00075   switch(direction){
00076   case 'n':
00077     dir = "nothern";
00078     break;
00079   case 'e':
00080     dir = "eastern";
00081     break;
00082   case 's':
00083     dir = "southern";
00084     break;
00085   case 'w':
00086     dir = "western";
00087     break;
00088   default:
00089     dir = "ERROR";
00090   }
00091   cout  « "The details of the "« dir «" most zipcode\n\t"
00092         « SSClass.fetch(SSClass.extremeCoord(state, direction));
00093 }
00094
00095 void quitProgram()
00096 {
00097     quit = true;
00098 }
00099
00100 void deleteRecord()
00101 {
00102     string field;
00103
00104     //ask user for pKey
00105     while(true)
00106     {
00107         cout « "Please enter the Zip Code of the Record to delete:" « endl;
00108         cin  » field;
00109         if(field.length() > 6 || field.length() < 1)
00110         {
00111             cout « "Invalid Zip Code entered. Please try again." « endl;
```

```
00112         }
00113         else
00114         {
00115             break;
00116         }
00117     }
00118
00119     cout « endl « "Searching for Record" « endl;
00120     int position = SSClass.binarySearchSS(field);
00121     if(position != -1)
00122     {
00123         cout « "Deleting Record from Sequence Set." « endl;
00124         if ( SSClass.deleteRecord( stoi( field ) ) )
00125         {
00126             cout « "Record deleted Successfully." « endl;
00127         }
00128         else
00129         {
00130             cout « "Error Deleting Record." « endl;
00131         }
00132     }
00133     else
00134     {
00135         cout « "Record not found in Sequence Set" « endl;
00136     }
00137 }
00138
00139 void searchForRecord()
00140 {
00141     string field;
00142
00143     //ask user for pKey
00144     while(true)
00145     {
00146         cout « "Please enter the Zip Code of the Record to find:" « endl;
00147         cin  » field;
00148         if(field.length() > 6 || field.length() < 1)
00149         {
00150             cout « "Invalid Zip Code entered. Please try again." « endl;
00151         }
00152         else
00153         {
00154             break;
00155         }
00156     }
00157
00158     cout « endl « "Searching for Record" « endl;
00159     int position = SSClass.binarySearchSS(field);
00160     if(position != -1)
00161     {
00162         cout « "Record found in Sequence Set." « endl;
00163         cout « "Displaying Record:" « endl « endl;
00164         cout « SSClass.fetch(field) « endl;
00165     }
00166     else
00167     {
00168         cout « "Record not found in Sequence Set" « endl;
00169     }
00170
00171 }
00172
00173 void addNewRecord()
00174 {
00175     string field;
00176     Record record;
00177
00178     //ask user for zip code
00179     while(true)
00180     {
00181         cout « "Please enter the Zip Code of the Record:" « endl;
00182         cin  » field;
00183         if(field.length() > 6 || field.length() < 1)
00184         {
00185             cout « "Invalid Zip Code entered. Please try again." « endl;
00186         }
00187         else
00188         {
00189             record.set_field("zip", field);
00190             break;
00191         }
00192     }
00193
00194     //ask user for city
00195     while(true)
00196     {
00197         cout « "Please enter the City of the Record: (use underscore _ as space)" « endl;
00198         cin  » field;
```

```
00199         if(field.length() > 31 || field.length() < 1)
00200         {
00201             cout « "Invalid City entered. Please try again." « endl;
00202         }
00203         else
00204         {
00205             for(int i = 0; i < field.length(); i++)
00206             {
00207                 if(field[i] == '_')
00208                     field[i] = ' ';
00209             }
00210             record.set_field("city", field);
00211             break;
00212         }
00213     }
00214
00215     //ask user for state
00216     while(true)
00217     {
00218         cout « "Please enter the State of the Record (two character format: MN):" « endl;
00219         cin  » field;
00220         if( field.length() != 2 )
00221         {
00222             cout « "Invalid State entered. Please try again." « endl;
00223         }
00224         else
00225         {
00226             record.set_field("state", field);
00227             break;
00228         }
00229     }
00230
00231     //ask user for county
00232     while(true)
00233     {
00234         cout « "Please enter the County of the Record:" « endl;
00235         cin  » field;
00236         if(field.length() > 38 || field.length() < 1)
00237         {
00238             cout « "Invalid County entered. Please try again." « endl;
00239         }
00240         else
00241         {
00242             record.set_field("county", field);
00243             break;
00244         }
00245     }
00246
00247     //ask user for longitude
00248     while(true)
00249     {
00250         cout « "Please enter the Longitude of the Record:" « endl;
00251         cin  » field;
00252         if(field.length() > 8 || field.length() < 1)
00253         {
00254             cout « "Invalid Longitude entered. Please try again." « endl;
00255         }
00256         else
00257         {
00258             record.set_field("long", field);
00259             break;
00260         }
00261     }
00262
00263     //ask user for latitude
00264     while(true)
00265     {
00266         cout « "Please enter the Latitude of the Record:" « endl;
00267         cin  » field;
00268         if(field.length() > 9 || field.length() < 1)
00269         {
00270             cout « "Invalid Latitude entered. Please try again." « endl;
00271         }
00272         else
00273         {
00274             record.set_field("lat", field);
00275             break;
00276         }
00277     }
00278
00279     cout « endl « "New Record Created." « endl;
00280     SSClass.addRecord( record );
00281     cout « "New Record added to Sequence Set." « endl;
00282 }
00283
00284 int main_menu()
00285 {
```

```
00286   int userResponce;
00287   while(true)
00288   {
00289       cout « "Please select an option:" « endl;
00290       cout « "1. Add a new Record" « endl;
00291       cout « "2. Search for and Display a Record" « endl;
00292       cout « "3. Delete a Record" « endl;
00293       cout « "4. Quit Program" « endl;
00294       cout « "5. Find the X-treme coordinate of a state" « endl;
00295       cout « "6. Testing" « endl;
00296       cin  » userResponce;
00297
00298       if(userResponce < 1 || userResponce > 6)
00299           cout « "Please enter a valid option" « endl;
00300       else
00301         return userResponce;
00302   }
00303 }
00304
00305 void SSDeleteAndAddRecordTester()
00306 {
00307   SequenceSet SSClass;
00308
00309   SSClass.deleteRecord(1008);
00310   SSClass.deleteRecord(1003);
00311   SSClass.deleteRecord(1004);
00312
00313   string zip = "563";
00314   string place = "Little Falls";
00315   string state = "MN";
00316   string county = "Morrison";
00317   string longitude = "-74.25";
00318   string latitude = "79.72";
00319   Record testRecord(zip, place, state, county, longitude, latitude);
00320   SSClass.addRecord(testRecord);
00321
00322   zip = "1024";
00323   Record testRecord2(zip, place, state, county, longitude, latitude);
00324   SSClass.addRecord(testRecord2);
00325
00326   zip = "1025";
00327   Record testRecord3(zip, place, state, county, longitude, latitude);
00328   SSClass.addRecord(testRecord3);
00329
00330   zip = "1051";
00331   Record testRecord4(zip, place, state, county, longitude, latitude);
00332   SSClass.addRecord(testRecord4);
00333
00334   zip = "1052";
00335   Record testRecord5(zip, place, state, county, longitude, latitude);
00336   SSClass.addRecord(testRecord5);
00337
00338   zip = "300";
00339   Record testRecord6(zip, place, state, county, longitude, latitude);
00340   SSClass.addRecord(testRecord6);
00341
00342   SSClass.rewriteSSFile();
00343 }
00344
00345 void nullblockTester(){
00346   Block * aBlock;
00347   ofstream sequenceSetFile;
00348   string fileName = "Sequence_Set.txt";
00349   sequenceSetFile.open(fileName);
00350   sequenceSetFile « "Hello File\n";
00351   sequenceSetFile.close();
00352
00353   string records[4] = {"501", "544", "1001", ""};
00354   string blockInfo = "   501   544  1001  1002";
00355
00356   //test block constructor
00357   Block anotherBlock(blockInfo);
00358   anotherBlock.write(fileName);
00359
00360   //test block search method
00361   string recordTest = "1002";
00362   aBlock = new Block(1);
00363   cout « "Return 1 if the record was found: " « aBlock->search( recordTest ) « endl;
00364
00365   recordTest = "103";
00366   aBlock->addRecord(recordTest);
00367
00368   recordTest = "103";
00369   aBlock->addRecord(recordTest);
00370
00371   recordTest = "544";
00372   aBlock->deleteRecord(recordTest);
```

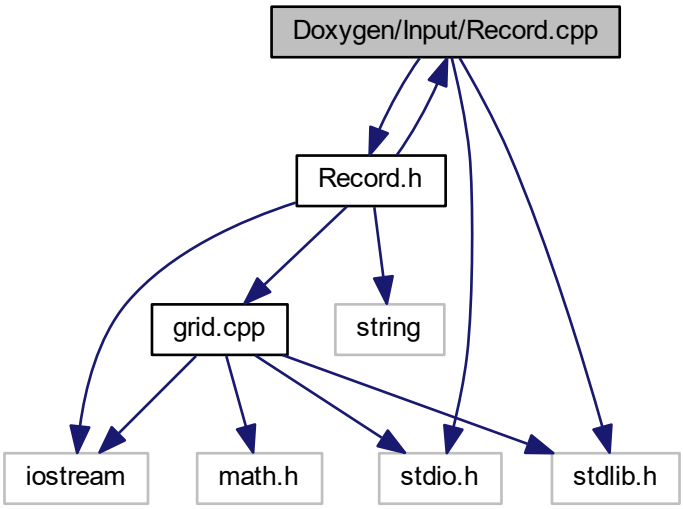
```
00373
00374   recordTest = "514";
00375   aBlock->deleteRecord(recordTest);
00376 }
00377
00378 void blockTester(){
00379   Block aBlock;
00380   ofstream sequenceSetFile;
00381   string fileName = "Sequence_Set.txt";
00382   sequenceSetFile.open(fileName);
00383   sequenceSetFile « "Hello File\n";
00384   sequenceSetFile.close();
00385
00386   string records[4] = {"501", "544", "1001", ""};
00387   string blockInfo = "   501   544  1001  1002";
00388
00389   //test block constructor
00390   Block anotherBlock(blockInfo);
00391   anotherBlock.write(fileName);
00392
00393   //test block search method
00394   string recordTest = "1002";
00395   cout « "Return 1 if the record was found: " « anotherBlock.search( recordTest ) « endl;
00396
00397   recordTest = "103";
00398   anotherBlock.addRecord(recordTest);
00399
00400   recordTest = "103";
00401   anotherBlock.addRecord(recordTest);
00402
00403   recordTest = "544";
00404   anotherBlock.deleteRecord(recordTest);
00405
00406   recordTest = "514";
00407   anotherBlock.deleteRecord(recordTest);
00408 }
00409
00410 void truncateTester(){
00411   Truncate t;
00412   Truncate t2(5);
00413   string str = "123456789AB";
00414
00415   cout « endl « "The String is " « str;
00416   cout « endl « "The String AS it is modified is " « t.modifyString(str);
00417   cout « endl « "The String IF it was modified is " « t2.truncatedString(str);
00418   cout « endl « "The String is " « str « endl;
00419 }
00420
00421 void recordTester(){
00422   //test default constructor
00423     Record testRecord;
00424   cout « "Default constructor record (should be empty):";
00425   testRecord.display();
00426   cout « endl;
00427
00428   //test fill record
00429   string zip = "56345";
00430   string place = "Little Falls";
00431   string state = "Minnesota";
00432   string county = "Morrison";
00433   string longitude = "-74.25";
00434   string latitude = "79.72";
00435
00436   cout « "Fill Record with : " « zip « " " « place « " " « state « " " « county « " " « longitude « "
      " « latitude;
00437
00438   testRecord.set_field( "z", zip );
00439   testRecord.set_field( "place", place );
00440   testRecord.set_field( "STATE", state );
00441   testRecord.set_field( "c", county );
00442   testRecord.set_field( "long", longitude );
00443   testRecord.set_field( "lat", latitude );
00444
00445   testRecord.display();
00446   cout « endl;
00447
00448   //test constructor 2
00449   float longitude_float = 74.25;
00450   float latitude_float = 79.72;
00451
00452   Record testRecord2(zip, place, state, county, longitude, latitude);
00453
00454   cout « "Constructor2 record (record should be full):";
00455   testRecord2.display();
00456
00457     //test constructor 3
00458   Grid grid_test(longitude_float, latitude_float);
```

```
00459
00460   Record testRecord3(zip, place, state, county, grid_test);
00461
00462   cout « "Constructor3 record (record should be full):";
00463   testRecord3.display();
00464
00465   //test display field
00466   cout « endl « "Test Display Field, display city:";
00467   testRecord3.display("CITY");
00468   cout « " expected: Little Falls" « endl;
00469
00470   cout « "Test Display Field, display state:";
00471   testRecord3.display("STATE");
00472   cout « " expected: Minnesota" « endl;
00473 }
```

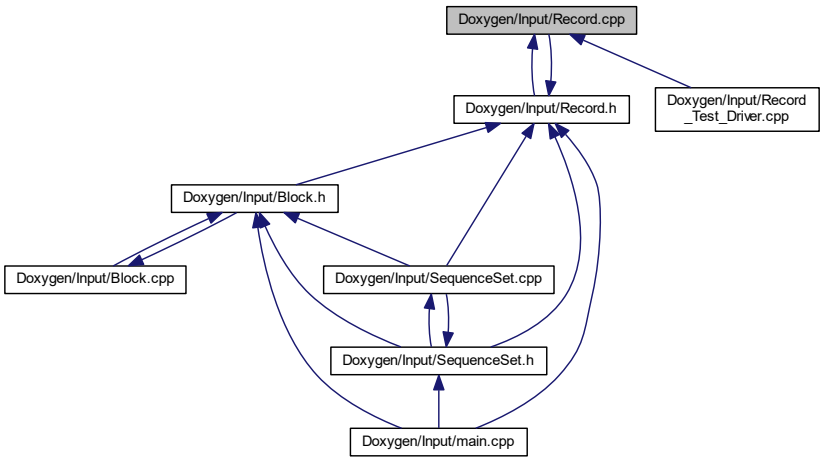
4.11 Doxygen/Input/Record.cpp File Reference

```
#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;
```

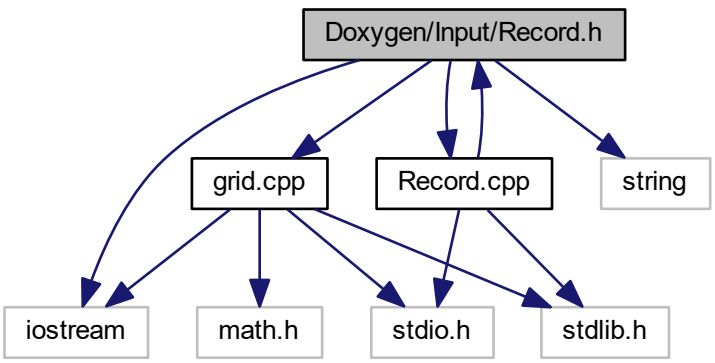
```
00064   place_name = _place_name;
00065   state = _state;
00066   county = _county;
00067   this -> set_longitude_latitude( lon, lat );
00068   if(DEBUG) {cout « "Made a filled record using string lat/longs.\n";}
00069 }
00070
00071 void Record::display()
00072 {
00073     if(DEBUG) {cout « "Displaying 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 }
```

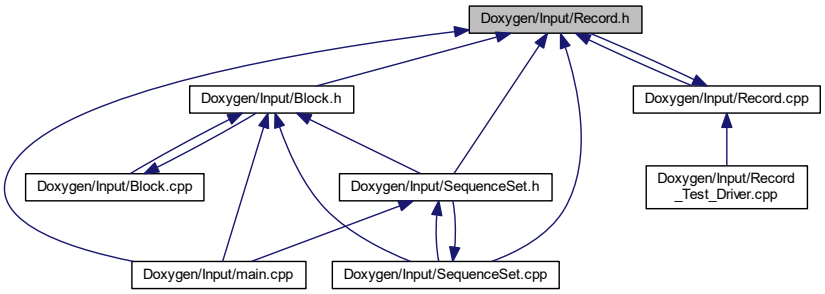
4.13 Doxygen/Input/Record.h File Reference

```
#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

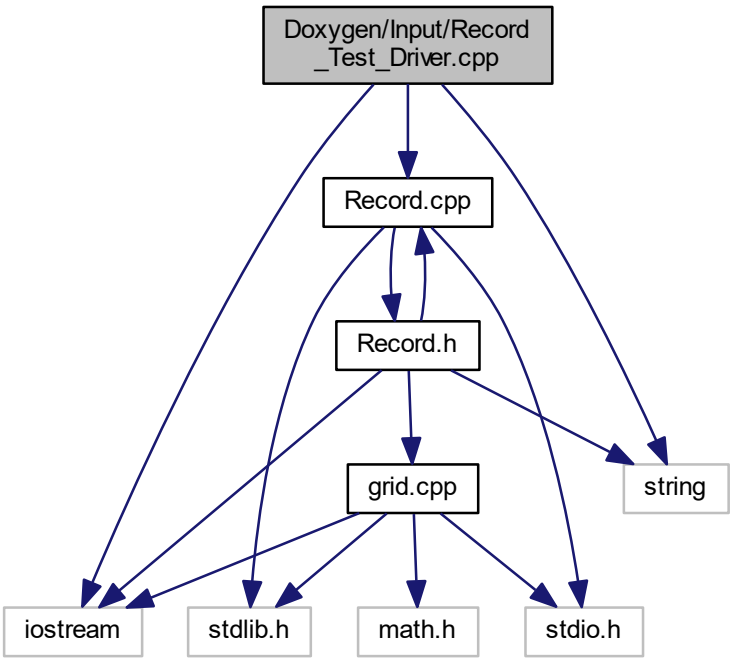
4.14 Record.h

```
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 Doxygen/Input/Record_Test_Driver.cpp File Reference

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

Include dependency graph for Record_Test_Driver.cpp:



Enumerations

- enum Field {
  Z, ZIP, CITY, P,
  PLACE_NAME, STATE, COUNTY, G,
  GRID }

Functions

- int main ()

4.15.1 Enumeration Type Documentation

4.15.1.1 Field

```
enum Field
```

Enumerator

Z	
ZIP	
CITY	
P	
PLACE_NAME	
STATE	
COUNTY	
G	
GRID	

Definition at line 9 of file Record_Test_Driver.cpp.

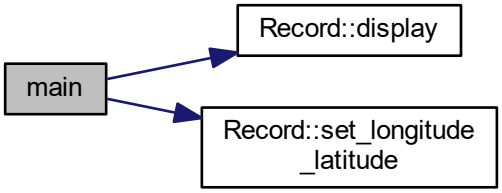
4.15.2 Function Documentation

4.15.2.1 main()

```
int main ( )
```

Definition at line 22 of file Record_Test_Driver.cpp.

Here is the call graph for this function:



4.16 Record_Test_Driver.cpp

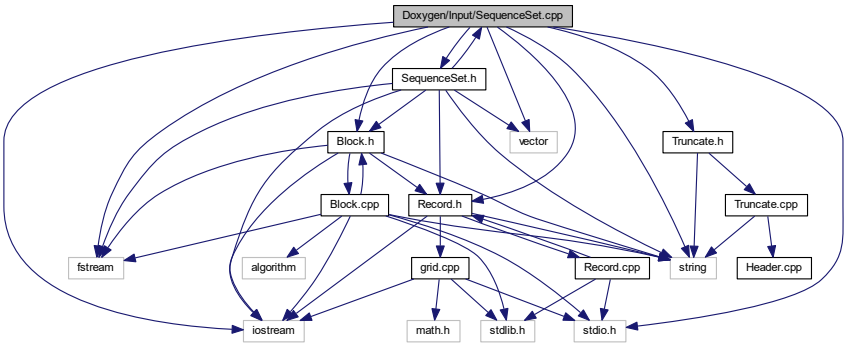
```
00001 //g++ -std=c++11  -o record_test Record_Test_Driver
00002
00003 #include "Record.cpp"
00004 #include<iostream>
00005 #include<string>
00006
00007 using namespace std;
00008
00009 enum Field
00010 {
00011     Z,
00012     ZIP,
00013     CITY,
```

```
00014     P,
00015     PLACE_NAME,
00016     STATE,
00017     COUNTY,
00018     G,
00019     GRID
00020 };
00021
00022 int main()
00023 {
00024   //test default constructor
00025     Record testRecord;
00026   cout « "Default constructor record:";
00027   testRecord.display();
00028
00029   //test fill record
00030   string zip = "56345";
00031   string place = "Little Falls";
00032   string state = "Minnesota";
00033   string county = "Morrison";
00034   float longitude = 74.25;
00035   float latitude = 79.72;
00036
00037   testRecord.set_zip_code( zip );
00038   testRecord.set_place_name( place );
00039   testRecord.set_state( state );
00040   testRecord.set_county( county );
00041   testRecord.set_longitude_latitude( longitude, latitude );
00042
00043   cout « "Filled Record:";
00044   testRecord.display();
00045
00046   //test constructor 2
00047   string longitude_string = "74.25";
00048   string latitude_string = "79.72";
00049
00050   Record testRecord2(zip, place, state, county, longitude_string, latitude_string);
00051
00052   cout « "Constructor2 record (long/lat are strings):";
00053   testRecord2.display();
00054
00055     //test constructor 3
00056   Grid grid_test(longitude, latitude);
00057
00058   Record testRecord3(zip, place, state, county, grid_test);
00059
00060   cout « "Constructor3 record (long/lat are gridPoint):";
00061   testRecord3.display();
00062
00063   cout « endl « "check enum:";
00064   testRecord3.display(CITY);
00065   cout « " expected: Little Falls" « endl;
00066
00067   testRecord3.display(STATE);
00068   cout « " expected: Minnesota" « endl;
00069
00070     return 0;
00071 }
```

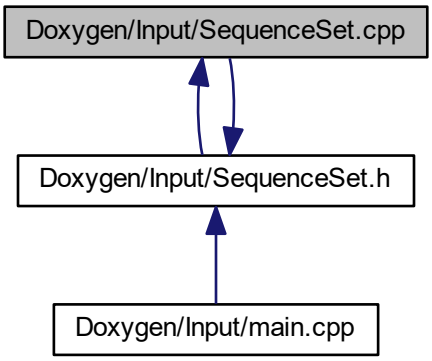
4.17 Doxygen/Input/SequenceSet.cpp File Reference

```
#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)

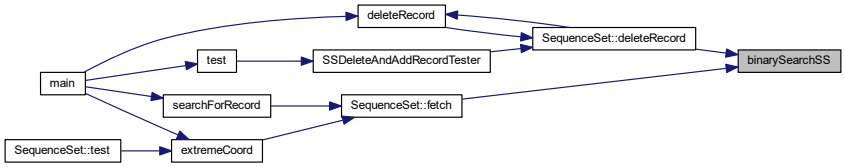
4.17.1 Function Documentation

4.17.1.1 binarySearchSS()

```
int binarySearchSS (
            const string arr[],
```

```
            string x,
            int n )
```

Here is the caller graph for this function:



4.18 SequenceSet.cpp

```
00001
00019 #include "SequenceSet.h"
00020 #include <iostream>
00021 #include "Truncate.h"
00022 #include "Record.h"
00023 #include "Block.h"
00024 #include "SequenceSet.h"
00025 #include <string>
00026 #include <fstream>
00027 #include <vector>
00028 #include <stdio.h>
00029
00030 using namespace std;
00031
00032 //binarySearch recycled from block
00033 int binarySearchSS(const string arr[], string x, int n);
00034
00035 SequenceSet::SequenceSet(){
00036   ofstream SSFile;
00037   SSFile.open(SSFileName);
00038   SSFile « "Sequence Set File\n";
00039   SSFile.close();
00040   recordCount = getRecordCount();
00041   fillIndex();
00042   Block * currentBlock = headBlock;
00043   blockCount = 0;
00044   for(unsigned long long i = 0; i < recordCount; i++){
00045     if(i%BLOCKFILLCOUNT == 0 && i != 0){
00046         if(DEBUG){cout « "Making a new block for the chain." « endl;}
00047       blockCount++;
00048       Block * newBlock = new Block(blockCount);
00049       currentBlock->setNextBlock(newBlock);
00050       newBlock->setPrevBlock(currentBlock);
00051       currentBlock = newBlock;
00052     }
00053     if(DEBUG){cout«"Passing "«to_string(pKeyIndex.at(i))«" into the add function."«endl;}
00054     currentBlock->addRecord(to_string(pKeyIndex.at(i)));
00055   }
00056   writeBlocks();
00057
00058   //reset the record avail list
00059   ofstream recordAvailList;
00060   recordAvailList.open(recordAvailListFileName);
00061   recordAvailList « "";
00062   recordAvailList.close();
00063   sKeyStateBuilder();
00064 }// End default constructor
00065
00066 unsigned long long SequenceSet::headerLength(string _fileName){
00067   fstream data;
00068   unsigned long long length = 0;
00069   unsigned long long L = 0;
00070   data.open(_fileName);
00071   string str;
00072
00073   if(DEBUG){cout « "String outside while loop, in headerLength: " « str « endl;}
00074   while(data.peek() != EOF){
00075     if(DEBUG && false){cout « "String in headerLength: " « str « endl;}
00076     getline(data, str);
00077     length += str.length();
```

```
00078     length++;
00079     if(str == HEADERENDSTRING){
00080       L = length;
00081       if(DEBUG){cout«"L defined: "« L «"\n";}
00082     }
00083   }
00084
00085   data.close();
00086
00087   return L;
00088 }// End headerLength
00089
00090 unsigned int SequenceSet::getRecordCount(){
00091     string fileName = DATAFILENAME;
00092     string field;
00093     string str = "";
00094     char c;
00095     fstream data;
00096     unsigned int recordCount = 0;
00097     data.open(fileName);
00098     getline(data, field); //Skip title
00099     while(data.peek() != ':' ){
00100         data.get(c);
00101         field += c;
00102         if(DEBUG && true){cout « "Char c: " « c « endl;}
00103     }
00104
00105     //This while is to skip non number values before approaching what to do with the values
00106     while(data.peek() < '0' || data.peek() > '9' ){
00107         data.get(c);
00108     }
00109     getline(data, str);
00110
00111     recordCount = stoi(str);
00112
00113     if(DEBUG) {cout « "String: " « str « "\nrecords: " « recordCount « endl;}
00114     if(field == "Records"){
00115         getline(data, str);
00116         recordCount = stoi(str);
00117         if(DEBUG){cout « "Record Count: " « recordCount « endl;}
00118     }
00119     data.close();
00120
00121     return recordCount;
00122 }// End getRecordCount
00123
00124 void SequenceSet::fillIndex(){
00125     string field;
00126     string str = "";
00127     char c;
00128     fstream data;
00129
00130     data.open("RecordOffsets.txt");
00131
00132     for(unsigned int i = 0; i < recordCount; i++){
00133         string recordData = "";
00134         getline(data, recordData);
00135         if(DEBUG){cout « "recordData: " « recordData « endl;}
00136         str = "";
00137         for(int j = 0; j < ZIPLENGTH; j++){
00138           str += recordData[j];
00139         }
00140         //index[i][0] = stoi(str);  //five chars of string
00141         pKeyIndex.push_back(stoi(str));
00142         if(DEBUG){cout « "String: " « str « endl;}
00143         if(DEBUG){cout « "pKeyIndex.at(i): " « pKeyIndex.at(i) «endl;}
00144         str = "";
00145         for(int j = ZIPLENGTH; j < recordData.length(); j++){
00146           str += recordData[j];
00147         }
00148         //index[i][1] = stoi(str);  //the rest of the string
00149         offsetIndex.push_back(stoi(str));
00150         if(DEBUG){cout « "String: " « str « endl;}
00151         if(DEBUG){cout « "offsetIndex.at("«i«"): " « offsetIndex.at(i) «endl;}
00152     }
00153     data.close();
00154 }// End fillIndex
00155
00156 string SequenceSet::fetch(string pKey){
00157   fstream data;
00158   data.open(DATAFILENAME);
00159   string returnString = "";
00160   for(int i = ZIPLENGTH - pKey.length(); i > 0; i--){
00161     if(DEBUG){cout « "For loop in fetch. i = " « i « endl;}
00162     returnString += " ";
00163   }
00164   returnString = pKey;
```

```
00165   returnString += " not found.\n";
00166
00167   int position;
00168   if(pKey != ""){
00169     position = binarySearchSS(pKey);
00170   }
00171   if(DEBUG) {cout « "Searching "« pKey « " returned: " « position « endl;}
00172   if(position>=0 && pKey != ""){
00173     data.seekg(offsetIndex.at(binarySearchSS(pKey)));
00174     getline(data, returnString);
00175   }
00176   data.close();
00177
00178   return returnString;
00179 }// End fetch with string
00180
00181 string SequenceSet::fetch(unsigned int pKey){
00182   return fetch(to_string(pKey));
00183 }// End fetch with int
00184
00185 void SequenceSet::makeRecordOffsets(string fileName){
00186     string zip = "      ";
00187     fstream data;
00188     ofstream index;
00189     string str;
00190     index.open("RecordOffsets.txt");
00191     unsigned long long offset = headerLength(fileName);
00192     data.open(fileName);
00193     data.seekg(offset);
00194
00195     if(DEBUG && false){cout « "String in makeRecordOffsets is: " « str « endl;}
00196     getline(data, str);
00197
00198     while(data.peek()!=EOF){
00199       if (DEBUG && false){cout « str « endl;}
00200     for(int i = 0; i < ZIPLENGTH; i++){
00201       zip[i] = str[i];
00202     }
00203     if(DEBUG && false){cout«zip« " is at " « offset «endl;}
00204     index « zip « offset « endl;
00205     getline(data, str);
00206     offset += str.length();
00207     offset++;
00208     }
00209
00210     data.close();
00211     index.close();
00212 }//End makeRecordOffsets
00213
00214
00215 int SequenceSet::binarySearchSS(string x)
00216 {
00217     //int int_arr[n];
00218     unsigned int n = recordCount;
00219     int int_string;
00220 /*
00221     //convert the records (array of strings) to array of int
00222     for (unsigned int i = 0; i < n; i++)
00223     {
00224     if(arr[i] != null_str)
00225           int_arr[i] = stoi(arr[i]);
00226     }
00227 */
00228     //convert string to find to int
00229     if(DEBUG){cout « "(stoi)ing this string: \"" « x « "\"\n";}
00230     try{
00231     int_string = stoi(x);    unsigned int l = 0 ;
00232     unsigned int r = n - 1;
00233     while (l <= r)
00234     {
00235       int m = l + (r - l) / 2;
00236           if(DEBUG) {cout « "mid: " « m «endl;}
00237
00238       //if(DEBUG) {cout « "comparing " « int_string « " and " « int_arr[m] «endl;}
00239       if(DEBUG) {cout « "comparing " « int_string « " and " « pKeyIndex.at(m) «endl;}
00240
00241         if ( pKeyIndex.at(m) == int_string ){
00242         if(DEBUG) {cout « "record found" «endl;}
00243         return m;
00244           }
00245
00246       // If x is greater, ignore left half
00247       if ( pKeyIndex.at(m) < int_string ){
00248         l = m + 1;
00249         if(DEBUG) {cout « "new l: " « l «endl;}
00250       }
00251
```

```
00252         // If x is smaller, ignore right half
00253       else{
00254         r = m - 1;
00255         if(DEBUG) {cout « "new r: " « l «endl;}
00256       }
00257     }
00258 }
00259     catch(...){cout « "ERROR (stoi)ING THIS STRING: \"" « x « "\"\n";}
00260
00261     return -1;
00262 }// End binarySearchSS
00263
00264 Record SequenceSet::fillRecord(string RecordString){
00265   string zip_code, place_name, state, county, latitude, longitude;
00266   int position = 0;
00267   if(DEBUG){cout  « "In fillRecord for Sequence Set Class\n\tRecordString: "
00268                   « RecordString « endl;}
00269   zip_code = "";
00270   for(auto i = 0; i < ZIPLENGTH    ; i++){
00271     if(RecordString[position] != ' '){
00272       zip_code += RecordString[position];
00273     }
00274     position++;
00275   }
00276
00277   place_name = "";
00278   for(int i = 0; i < 31/*Length of place name*/; i++){
00279     if(RecordString[position] != ' '){
00280       place_name += RecordString[position];
00281     }
00282     position++;
00283   }
00284
00285   state = "";
00286   for(int i = 0; i < 2/*Length of state*/; i++){
00287     if(RecordString[position] != ' '){
00288       state += RecordString[position];
00289     }
00290     position++;
00291   }
00292
00293   county = "";
00294   for(int i = 0; i < 38/*Length of county*/; i++){
00295     if(RecordString[position] != ' '){
00296       county += RecordString[position];
00297     }
00298     position++;
00299   }
00300
00301   latitude = "";
00302   for(int i = 0; i < 9/*Length of latitude*/; i++){
00303     if(RecordString[position] != ' '){
00304       latitude += RecordString[position];
00305     }
00306     position++;
00307   }
00308
00309   longitude = "";
00310   for(int i = 0; i < 8/*Length of longitude*/; i++){
00311     if(RecordString[position] != ' '){
00312       longitude += RecordString[position];
00313     }
00314     position++;
00315   }
00316   if(DEBUG){cout  « "\tRecordElements: " « "\n\t\t"
00317                   « zip_code « "\n\t\t" « place_name « "\n\t\t"
00318                       « state « "\n\t\t" « county « "\n\t\t"
00319                   « latitude « "\n\t\t" « longitude « endl;}
00320
00321   Record returnRecord(zip_code, place_name, state, county, latitude, longitude);
00322
00323   if(DEBUG){returnRecord.display();}
00324
00325   return returnRecord;
00326 }// End fillRecord
00327
00328 void SequenceSet::writeBlocks(){
00329   Block * currentBlock = headBlock;
00330   for(auto i = 0; i < blockCount; i ++){
00331     if(DEBUG){cout « "Writing block "« i «" from the chain." « endl;}
00332     currentBlock->write(SSFileName);
00333     currentBlock = currentBlock->getNextBlock();
00334   }
00335 }// End writeBlocks
00336
00337 void SequenceSet::fillRecordBlock(unsigned long long blockID){
00338   string str, zip, passed;
```

```
00339   Block * currentBlock = headBlock;
00340   for(auto i = 0; i < blockID; i++){
00341     currentBlock = currentBlock->getNextBlock();
00342   }
00343
00344   currentBlock->getRecords(recordBlock);
00345   for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00346     passed = fetch(recordBlock[i].get_field("ZIP"));
00347     if(DEBUG){
00348       cout  « "\n*****************************************"
00349             « "\nString passed to fill record: " « passed « endl;
00350     }
00351     if(passed != " not found.\n" && passed != " not found."){
00352       recordBlock[i] = fillRecord(passed);
00353       if(DEBUG){recordBlock[i].display();}
00354     }
00355   }
00356 }// End fillRecordBlock
00357
00358 void SequenceSet::addBlockStateKey(unsigned long long blockID){
00359   fillRecordBlock(blockID);
00360   Block * currentBlock = headBlock;
00361
00362   for(auto i = 0; i < blockID; i++){
00363     currentBlock = currentBlock->getNextBlock();
00364   }
00365
00366   for(auto i = 0; i < currentBlock->getRecordCount(); i++){
00367     string state = recordBlock[i].get_field("state");
00368
00369     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00370       string state = recordBlock[i].get_field("state");
00371
00372       if(state != ""){
00373     bool stateFound = false;
00374     unsigned int index = 0;
00375
00376     if(stateZips.size() == 0){
00377             vector <string> newRow;
00378             newRow.push_back(state);
00379             stateZips.push_back(newRow);
00380     }
00381
00382     while(index < stateZips.size() && !stateFound){
00383         if(stateZips[index].at(0) == state){
00384             if(DEBUG){cout « "Found " « state « " at index = " « index « endl;}
00385             stateFound = true;
00386         }
00387         else{index++;}
00388     }
00389
00390     if(!stateFound){
00391         if(DEBUG){cout « state«" not found.\n";}
00392             vector <string> newRow;
00393             newRow.push_back(state);
00394             stateZips.push_back(newRow);
00395         if(DEBUG){cout « stateZips[index].at(0)«" pushed successfully.\n";}
00396         if(DEBUG){
00397             stateZips[index].push_back(":)");
00398             cout « "Pushing a smily :)\n";
00399             cout « stateZips[index].at(1) « endl;
00400             stateZips[index].pop_back();
00401         }
00402     }
00403
00404     if(DEBUG){cout « "Pushing " « recordBlock[i].get_field("zip") «" to "« index «" column.\n";}
00405     stateZips[index].push_back(recordBlock[i].get_field("zip"));
00406     //if(DEBUG){cout « stateZips[index].at(stateZips[index].size())«" pushed successfully.\n";}
00407
00408
00409     if(DEBUG){cout « stateZips[index].at(0) « ": "
00410         « stateZips[index].at(stateZips[index].size()-1) « endl;}
00411       }
00412     }
00413   }
00414 }// End addBlockStateKey
00415
00416
00417 bool SequenceSet::deleteRecord(int pKey)
00418 {
00419   //search if the record is in the sequence set
00420   int position = binarySearchSS( to_string(pKey) );
00421   if(DEBUG) {cout « "Searching for "« pKey « " returned: " « position « endl;}
00422   if(position == -1){
00423     cout « "Record does not exist in Sequence Set." « endl;
00424     return false;
00425   }
```

```
00426   else{
00427     //add deleted record offset to avail list
00428     string strTemp = "";
00429     string newString = "";
00430     fstream recordAvailListIn;
00431     recordAvailListIn.open(recordAvailListFileName);
00432     while(recordAvailListIn.peek() != EOF){
00433         strTemp += recordAvailListIn.get();
00434         if(DEBUG){cout « strTemp « endl;}
00435     }
00436     newString = to_string( offsetIndex.at(position) ) + "/" + to_string( position ) + "\n" + strTemp;
00437     if(DEBUG){cout « newString « " result" « endl;}
00438     recordAvailListIn.close();
00439
00440     ofstream recordAvailList;
00441     recordAvailList.open(recordAvailListFileName);
00442     recordAvailList « newString;
00443     recordAvailList.close();
00444
00445     //delete record from us_postal_codes.txt
00446     fstream usPostalCodes;
00447     usPostalCodes.open("us_postal_codes.txt");
00448     usPostalCodes.seekg(offsetIndex.at(position));
00449     for(int i = 0; i < 94; i++){ //94 is the length of record
00450       usPostalCodes « " ";
00451     }
00452     usPostalCodes.close();
00453
00454     //delete record in index vector
00455     pKeyIndex.erase(pKeyIndex.begin() + position);
00456     offsetIndex.erase(offsetIndex.begin() + position);
00457       if(DEBUG) {position = binarySearchSS( to_string(pKey) );}
00458     if(DEBUG) {cout « "Deleted record in index vector. Researching for "« pKey « " returned: " «
    position « endl;}
00459     recordCount--; //decrement the total record count
00460
00461     //delete record in linked list of blocks
00462     Block * currentBlock = headBlock;
00463     for(auto i = 0; i < blockCount; i ++){
00464       if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
00465       if( pKey <= currentBlock->getLastRecordPKey() ){
00466         currentBlock->deleteRecord( to_string(pKey) );
00467         break;
00468       }
00469       else{
00470         currentBlock = currentBlock->getNextBlock();
00471       }
00472     }
00473
00474     //merge blocks if needed
00475     if( currentBlock->getRecordCount() < RECORDSPERBLOCK / 2 ){
00476       //check next block to see if it can merge
00477       if( (currentBlock->getNextBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00478         currentBlock->getRecords( recordBlock ); //get the pkeys
00479         for(int i = 0; i < currentBlock->getRecordCount(); i++){
00480           (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00481           currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00482         }
00483           //add the pointer to the current block to the avail vector
00484           blockAvailList.push_back( currentBlock );
00485           //change the pointers to avoid the empty block
00486           currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00487           currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00488           blockCount--;
00489       }
00490       //check if previous block can merge
00491       else if( (currentBlock->getPreviousBlock())->getRecordCount() == RECORDSPERBLOCK / 2 ){
00492         currentBlock->getRecords( recordBlock ); //get the pkeys
00493         for(int i = 0; i < currentBlock->getRecordCount(); i++){
00494           (currentBlock->getPreviousBlock())->addRecord(recordBlock[i].get_field("zip"));
00495           currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00496         }
00497           //add the pointer to the current block to the avail vector
00498           blockAvailList.push_back( currentBlock );
00499           //change the pointers to avoid the empty block
00500           currentBlock->getPreviousBlock()->setNextBlock( currentBlock->getNextBlock() );
00501           currentBlock->getNextBlock()->setPrevBlock( currentBlock->getPreviousBlock() );
00502           blockCount--;
00503       }
00504       //check if next block can redistribute
00505       else if( (currentBlock->getNextBlock())->getRecordCount() > RECORDSPERBLOCK / 2 ){
00506         (currentBlock->getNextBlock())->getRecords( recordBlock ); //get the pkeys
00507         currentBlock->addRecord( recordBlock[0].get_field("zip") );
00508         (currentBlock->getNextBlock())->deleteRecord( recordBlock[0].get_field("zip") );
00509       }
00510     }
00511     rewriteSSFile();
```

```
00512     return true;
00513   }
00514 }// End deleteRecord
00515
00516 string SequenceSet::extremeCoord(string state, char direction)
00517 {
00518   direction = toupper(direction);
00519     float extremePoint = 0;
00520     string zip = "";
00521   for(int i = 0; i < 2; i++){
00522     zip+=toupper(state[i]);
00523   }
00524   state = zip;
00525     zip = "";
00526   Record currentRecord;
00527   string str = state;
00528
00529   bool found = false;
00530   unsigned int index = 0;
00531   while(index < stateZips.size() - 1 && !found){
00532     if(stateZips[index][0] == str){found = true;}
00533     else{index++;}
00534   }
00535   currentRecord = fillRecord(fetch(stateZips[index][1]));
00536
00537     switch(direction)
00538     {
00539         case 'N':
00540         {
00541       extremePoint = stof(currentRecord.get_field("Lat"));
00542       zip = currentRecord.get_field("zip");
00543             for(int i = 1; i < stateZips[index].size(); i++)
00544             {
00545                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00546                 if(extremePoint < stof(currentRecord.get_field("Lat")))
00547                 {
00548           zip = currentRecord.get_field("zip");
00549                     extremePoint = stof(currentRecord.get_field("Lat"));
00550                 }
00551             }
00552         }
00553         break;
00554
00555         case 'E':
00556         {
00557             extremePoint = stof(currentRecord.get_field("Long"));
00558             zip = currentRecord.get_field("zip");
00559             for(int i = 1; i < stateZips[index].size(); i++)
00560             {
00561                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00562                 if(extremePoint < stof(currentRecord.get_field("Long")))
00563                 {
00564           zip = currentRecord.get_field("zip");
00565                     extremePoint = stof(currentRecord.get_field("Long"));
00566                 }
00567             }
00568         }
00569         break;
00570
00571         case 'S':
00572         {
00573             extremePoint = stof(currentRecord.get_field("Lat"));
00574             zip = currentRecord.get_field("zip");
00575             for(int i = 1; i < stateZips[index].size(); i++)
00576             {
00577                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00578                 if(extremePoint > stof(currentRecord.get_field("Lat")))
00579                 {
00580           zip = currentRecord.get_field("zip");
00581                     extremePoint = stof(currentRecord.get_field("Lat"));
00582                 }
00583             }
00584         }
00585         break;
00586
00587         case 'W':
00588         {
00589             extremePoint = stof(currentRecord.get_field("Long"));
00590             zip = currentRecord.get_field("zip");
00591             for(int i = 1; i < stateZips[index].size(); i++)
00592             {
00593                 currentRecord = fillRecord(fetch(stateZips[index][i]));
00594                 if(extremePoint > stof(currentRecord.get_field("Long")))
00595                 {
00596           zip = currentRecord.get_field("zip");
00597                     extremePoint = stof(currentRecord.get_field("Long"));
00598                 }
```

```
00599             }
00600         }
00601         break;
00602
00603     default:
00604     {
00605       cout « "UNDEFINED OPTION\n";
00606     }
00607     }
00608     return zip;
00609 }// End extremeCoord
00610
00611 int SequenceSet::test(){
00612     string field;
00613     string str = "";
00614     char c;
00615     fstream data;
00616
00617     int randomRecord = rand() % recordCount;
00618     //cout « "Retrieving record: " « index[randomRecord][0] « endl;
00619     cout « "Retrieving record: " « pKeyIndex.at(randomRecord) « endl;
00620     data.open(DATAFILENAME);
00621     //data.seekg(index[randomRecord][1]);
00622     data.seekg(offsetIndex.at(randomRecord));
00623     getline(data, str);
00624     cout « str « endl;
00625
00626     cout « fetch(1721) « endl;
00627     fillRecordBlock(88);
00628
00629     for(auto i = 0; i < RECORDSPERBLOCK; i++){
00630       if(DEBUG){cout «"\n**********************************\n";}
00631       recordBlock[i].display();
00632     }
00633
00634     sKeyStateBuilder();
00635
00636     unsigned int index = 0;
00637     unsigned int record = 1;
00638     Record currentRecord;
00639
00640     str = "MN";
00641     bool found = false;
00642     while(index < stateZips.size() && !found){
00643       if(stateZips[index][0] == str){found = true;}
00644       else{index++;}
00645     }
00646
00647     while(record < stateZips[index].size()){
00648       str = fetch(stateZips[index][record]);
00649       cout « str « endl;
00650       currentRecord = fillRecord(str);
00651       currentRecord.display();
00652       record++;
00653     }
00654
00655     cout « extremeCoord(str, 'n') « endl;
00656
00657     return 0;
00658 }// End test
00659
00660
00661 void SequenceSet::sKeyStateBuilder(){
00662     if(DEBUG){cout « "Building sKeys for states.\n";}
00663     Block * currentBlock = headBlock;
00664     unsigned int index = 0;
00665     while(currentBlock!=NULL){
00666       addBlockStateKey(index);
00667       currentBlock = currentBlock->getNextBlock();
00668       index++;
00669     }
00670 }//End sKeyStateBuilder
00671
00672 void SequenceSet::addRecord(Record record)
00673 {
00674     //search record in linked list of blocks
00675     Block * currentBlock = headBlock;
00676     for(auto i = 0; i < blockCount; i ++){
00677         if(DEBUG){cout « "Searching block "« i «" from the chain." « endl;}
00678         if( stoi( record.get_field("zip") ) <= currentBlock->getLastRecordPKey() ){ //find the right
      block
00679       if(currentBlock->getRecordCount() == RECORDSPERBLOCK){ //if the block is full, do block
      splitting
00680         if( !blockAvailList.empty() ){  //if there exists a current empty block
00681           Block* tempBlockPtr = blockAvailList.back(); //get the pointer to the empty block
00682           blockAvailList.pop_back();  //delete the pointer from the avail list
00683           //add the relative block to the linked list
```

```
00684           tempBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00685           tempBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
00686           (currentBlock->getNextBlock())->setPrevBlock(tempBlockPtr);
00687           currentBlock->setNextBlock(tempBlockPtr);
00688           //split the data into the new block number
00689           currentBlock->getRecords( recordBlock ); //get the pkeys
00690           for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00691             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00692             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00693           }
00694           //add the new record to the block
00695           currentBlock->addRecord( record.get_field("zip") );
00696           blockCount++;
00697           break; //stop searching through linked list of blocks
00698         }
00699         else{  //if a current empty block doesn't exist, create a new block......
00700           Block* newBlockPtr = new Block;
00701           newBlockPtr->setRBN(blockCount);
00702           newBlockPtr->setNextBlock( currentBlock->getNextBlock() );
00703           newBlockPtr->setPrevBlock( (currentBlock->getNextBlock())->getPreviousBlock() );
00704           (currentBlock->getNextBlock())->setPrevBlock(newBlockPtr);
00705           currentBlock->setNextBlock(newBlockPtr);
00706           //split the data into the new block number
00707           currentBlock->getRecords( recordBlock ); //get the pkeys
00708           for(int i = RECORDSPERBLOCK / 2; i < RECORDSPERBLOCK; i++){
00709             (currentBlock->getNextBlock())->addRecord(recordBlock[i].get_field("zip"));
00710             currentBlock->deleteRecord( recordBlock[i].get_field("zip") );
00711           }
00712           //add the new record to the block
00713           currentBlock->addRecord( record.get_field("zip") );
00714           blockCount++;
00715           break; //stop searching through linked list of blocks
00716         }
00717       }
00718       else{
00719         currentBlock->addRecord( record.get_field("zip") );
00720       }
00721             break; //stop searching through linked list of blocks
00722         }
00723         else{
00724             currentBlock = currentBlock->getNextBlock();
00725         }
00726     }
00727
00728   //add record to us_postal_codes.txt
00729   fstream recordAvailList;
00730   string str = "";
00731   string strTemp = "";
00732   string offset = "";
00733   string position = "";
00734   recordAvailList.open(recordAvailListFileName);
00735   if( recordAvailList.peek() != EOF ){ //if recordAvailList is not empty
00736     fstream usPostalCodes;
00737     usPostalCodes.open("us_postal_codes.txt");
00738     getline(recordAvailList, str); //get the offset and vector position from avail list
00739     int i = 0;
00740     while( str[i] != '/' ){ //parse the offset from the string
00741         offset += str[i];
00742         if(DEBUG){cout << offset << endl;}
00743         i++;
00744     }
00745     i++;
00746     while( i < str.length()  ){ //parse the position from the string
00747         position += str[i];
00748         if(DEBUG){cout << position << endl;}
00749         i++;
00750     }
00751     writeToTxt(record, offset, "us_postal_codes.txt");
00752     usPostalCodes.close();
00753     recordAvailList.close();
00754     //update the recordAvailList
00755     recordAvailList.open(recordAvailListFileName);
00756     str += "\n";
00757     if(DEBUG){cout << str << " str to delete" << endl;}
00758     while(recordAvailList.peek() != EOF){
00759         strTemp += recordAvailList.get();
00760         if(DEBUG){cout << strTemp << endl;}
00761         if(strTemp == str){
00762             strTemp = "";
00763         }
00764     }
00765     recordAvailList.close();
00766     remove("availRecordList.txt");
00767     ofstream recordAvailListOut;
00768     recordAvailListOut.open(recordAvailListFileName, ios::app);
00769     if(DEBUG){cout << strTemp << " result" << endl;}
00770     recordAvailListOut << strTemp;
```

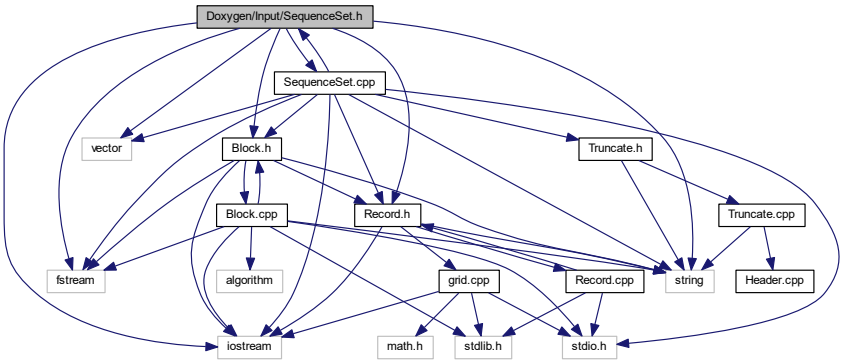
```
00771     recordAvailListOut.close();
00772     //add record to index vector
00773     if(DEBUG){for(int i=0; i<20; ++i)std::cout « pKeyIndex[i] « ' ';}
00774     pKeyIndex.insert(pKeyIndex.begin() + stoi( position ), stoi( record.get_field("zip") ) );
00775     offsetIndex.insert(offsetIndex.begin() + stoi( position ), stoi( offset ) );
00776     cout «endl;
00777     if(DEBUG){for(int i=0; i<20; ++i)std::cout « pKeyIndex[i] « ' ';}
00778   }
00779   else{ //if recordAvailList is empty
00780     unsigned int nextOffset = offsetIndex.back() + 95;//95 is record length+1
00781     if(DEBUG){cout « nextOffset « " nextoffset" « endl;}
00782     pKeyIndex.push_back( stoi( record.get_field("zip") ) );
00783     offsetIndex.push_back( nextOffset );
00784     writeToTxt(record, to_string( nextOffset ), "us_postal_codes.txt");
00785     ofstream usPostalCodes;
00786     usPostalCodes.open("us_postal_codes.txt", ios::app);
00787     usPostalCodes « endl;
00788     usPostalCodes.close();
00789   }
00790
00791   rewriteSSFile();
00792 }// End addRecord
00793
00794 void SequenceSet::rewriteSSFile()
00795 {
00796     //rewrite the squence set file with missing record
00797     remove("Sequence_Set.txt");
00798     ofstream SSFile;
00799     SSFile.open(SSFileName);
00800     SSFile « "Sequence Set File\n";
00801     SSFile.close();
00802     writeBlocks();
00803 }//End rewriteSSFile
00804
00805 //write the record to the postal codes file
00806 void SequenceSet::writeToTxt(Record record, string offset, string _fileName)
00807 {
00808     fstream data;
00809     data.open(_fileName);
00810     data.seekg( stoi( offset ) );
00811
00812     string dataString = "";
00813     string totalString = "";
00814
00815     dataString = record.get_field("Zip");
00816     int fieldLength = 6;
00817     for(int i = 0; i < fieldLength - dataString.length(); i++){
00818         totalString += " ";
00819     }
00820     totalString += dataString;
00821
00822     dataString = record.get_field("city");
00823     fieldLength = 31;
00824     totalString += dataString;
00825     for(int i = 0; i < fieldLength - dataString.length(); i++){
00826         totalString += " ";
00827     }
00828
00829     dataString = record.get_field("state");
00830     totalString += dataString;
00831
00832     dataString = record.get_field("county");
00833     fieldLength = 38;
00834     totalString += dataString;
00835     for(int i = 0; i < fieldLength - dataString.length(); i++){
00836         totalString += " ";
00837     }
00838
00839     dataString = record.get_field("long");
00840     while(dataString.length() > 7){
00841         dataString.pop_back();
00842     }
00843     fieldLength = 8;
00844     for(int i = 0; i < fieldLength - dataString.length(); i++){
00845         totalString += " ";
00846     }
00847     totalString += dataString;
00848
00849     dataString = record.get_field("lat");
00850     fieldLength = 9;
00851     while(dataString.length() > 8){
00852         dataString.pop_back();
00853     }
00854     for(int i = 0; i < fieldLength - dataString.length(); i++){
00855         totalString += " ";
00856     }
00857     totalString += dataString;
```

```
00858
00859     data « totalString;
00860
00861     data.close();
00862 }// End writeToTxt
```

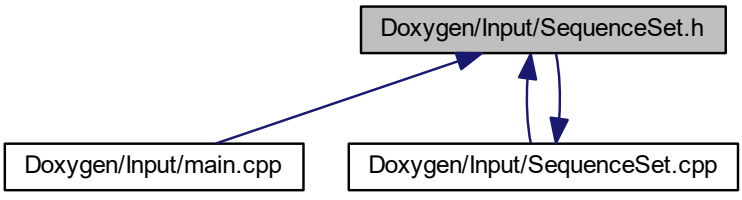
4.19 Doxygen/Input/SequenceSet.h File Reference

```
#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

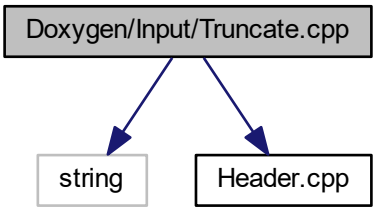
4.20 SequenceSet.h

```
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
00065
00066 public:
00072     SequenceSet();
00073
00079     void makeRecordOffsets(string fileName);
00080
00086     void fillIndex();
00087
00092     void fillRecordBlock(unsigned long long blockID);
00093
00098     void writeBlocks();
00099
00105     Record fillRecord(string RecordString);
00106
00112     unsigned int getRecordCount();
00113
00118     string fetch(string pKey);
00119
00124     string fetch(unsigned int pKey);
00125
00131     string extremeCoord(string, char);
00132
00137     bool deleteRecord(int pKey);
00138
00143     void addRecord(Record record);
00144     void rewriteSSFile();
00150     void writeToTxt(Record, string, string);
00151
00156     int binarySearchSS(string x);
00157
00162     int test();
00163 };
00164
00165 #include "SequenceSet.cpp"
00166
00167 #endif
00168
```

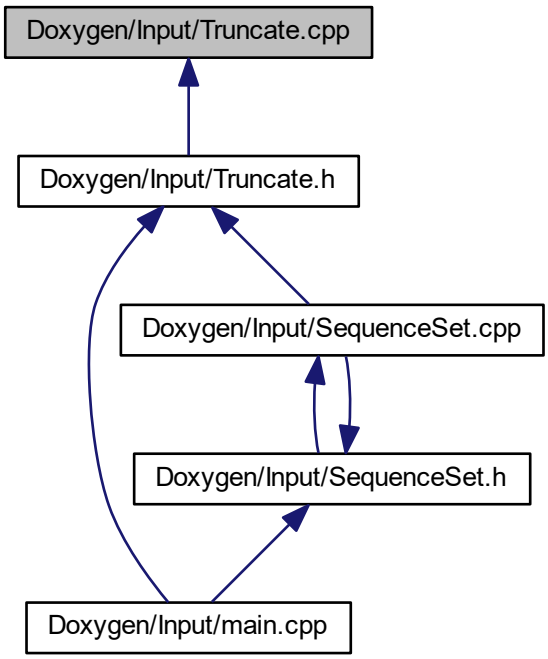
4.21 Doxygen/Input/Truncate.cpp File Reference

```
#include <string>
#include "Header.cpp"
```

Include dependency graph for Truncate.cpp:



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



4.21.1 Detailed Description

Author

Christenson, Mark

Definition in file Truncate.cpp.

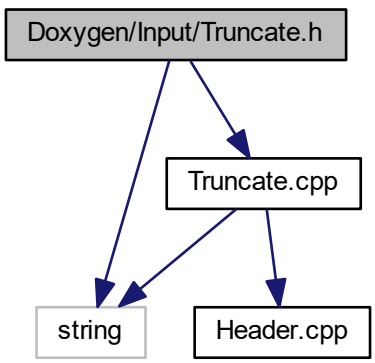
4.22 Truncate.cpp

```
00001
00006 //#include "Truncate.h"
00007 #include <string>
00008 #include "Header.cpp"
00009
00010 Truncate::Truncate(){
00011     if (DEBUG)
00012     {
00013         cout « "Truncate object made";
00014     }
00015 }
00016 Truncate::Truncate(int _size){
00017     maxLength = _size;
00018 }
00019
00020 string Truncate::truncatedString(string _string) {
00021     string newStr = _string;
00022     newStr.resize(maxLength);
00023
00024     return newStr;
00025 }
00026
00027 string Truncate::modifyString(string & _originalStr) {
00028     _originalStr.resize(maxLength);
00029
00030     return _originalStr;
00031 }
```

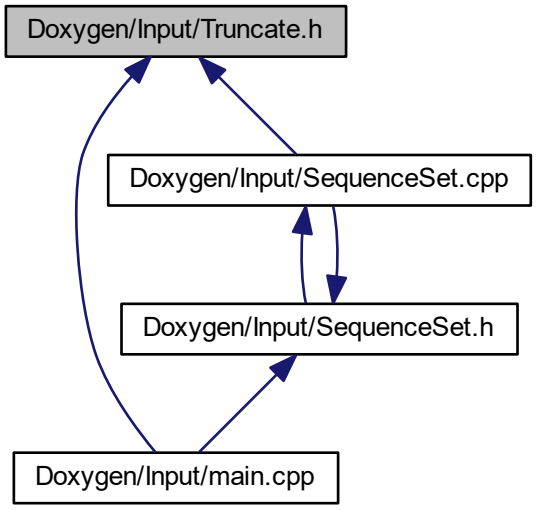
4.23 Doxygen/Input/Truncate.h File Reference

```
#include <string>
#include "Truncate.cpp"
```

Include dependency graph for Truncate.h:



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



Classes

- class Truncate

4.24 Truncate.h

```
00001 #ifndef TRUNCATE_H
00002 #define TRUNCATE_H
00003 #include <string>
00004
00005 using namespace std;
00006
00007 class Truncate {
00008 public:
00012     Truncate();
00016     Truncate(int);
00020     string modifyString(string&);
00025     string truncatedString(string);
00026 private:
00027     int maxLength = 10;
00028 };
00029
00030 #include "Truncate.cpp"
00031
00032 #endif
```

Index