Project 3 Zip Code - Team 4

1.0

Generated by Doxygen 1.14.0

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

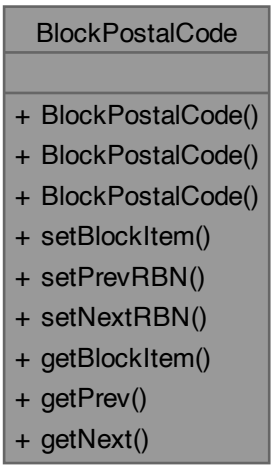
Class Documentation

3.1 BlockPostalCode Class Reference

Represents one block that stores a HeaderRecordPostalCodeItem.

`#include <BlockPostalCode.h>`

Collaboration diagram for BlockPostalCode:



Public Member Functions

- BlockPostalCode ()

  Default constructor.
- BlockPostalCode (const HeaderRecordPostalCodeItem &item)

  Constructor that sets the data item for this block.

- BlockPostalCode (const HeaderRecordPostalCodeItem &item, BlockPostalCode *prevBlock, BlockPostalCode *nextBlock)

  Constructor that sets data and predecessor/successor links.
- void setBlockItem (const HeaderRecordPostalCodeItem &item)

  Sets the header record stored in this block.
- void setPrevRBN (BlockPostalCode *prevBlock)

  Sets the predecessor block link.
- void setNextRBN (BlockPostalCode *nextBlock)

  Sets the successor block link.
- HeaderRecordPostalCodeItem getBlockItem () const

  Retrieves the header record stored in this block.
- BlockPostalCode * getPrev () const

  Gets the predecessor block pointer.
- BlockPostalCode * getNext () const

  Gets the successor block pointer.

3.1.1 Detailed Description

Represents one block that stores a HeaderRecordPostalCodeItem.

The block contains:

- The postal-code header record stored as `data`
- A predecessor pointer (`prev`) connecting to the previous block
- A successor pointer (`next`) connecting to the next block

These act as "predecessor & successor Relative Block Number links."

Definition at line 27 of file BlockPostalCode.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BlockPostalCode() [1/3]

```
BlockPostalCode::BlockPostalCode ()
```

Default constructor.

Initializes an empty block with null links.

Definition at line 12 of file BlockPostalCode.cpp.

3.1.2.2 BlockPostalCode() [2/3]

```
BlockPostalCode::BlockPostalCode (
            const HeaderRecordPostalCodeItem & item)
```

Constructor that sets the data item for this block.

Parameters

item	The HeaderRecordPostalCodeItem to store in the block.

Definition at line 18 of file BlockPostalCode.cpp.

3.1.2.3 BlockPostalCode() [3/3]

```
BlockPostalCode::BlockPostalCode (
            const HeaderRecordPostalCodeItem & item,
            BlockPostalCode * prevBlock,
            BlockPostalCode * nextBlock)
```

Constructor that sets data and predecessor/successor links.

Parameters

item	The header record stored in this block.
prevBlock	Pointer to the previous block.
nextBlock	Pointer to the next block.

Definition at line 26 of file BlockPostalCode.cpp.

References BlockPostalCode().

3.1.3 Member Function Documentation

3.1.3.1 setBlockItem()

```
void BlockPostalCode::setBlockItem (
            const HeaderRecordPostalCodeItem & item)
```

Sets the header record stored in this block.

Parameters

item	The new HeaderRecordPostalCodeItem to store.

Definition at line 32 of file BlockPostalCode.cpp.

3.1.3.2 setPrevRBN()

```
void BlockPostalCode::setPrevRBN (
            BlockPostalCode * prevBlock)
```

Sets the predecessor block link.

Parameters

prevBlock	Pointer to the previous block.

Definition at line 41 of file BlockPostalCode.cpp.

References BlockPostalCode().

3.1.3.3 setNextRBN()

```
void BlockPostalCode::setNextRBN (
            BlockPostalCode * nextBlock)
```

Sets the successor block link.

Parameters

nextBlock	Pointer to the next block.

Definition at line 50 of file BlockPostalCode.cpp.

References BlockPostalCode().

3.1.3.4 getBlockItem()

```
HeaderRecordPostalCodeItem BlockPostalCode::getBlockItem () const
```

Retrieves the header record stored in this block.

Returns

The HeaderRecordPostalCodeItem stored inside the block.

Definition at line 59 of file BlockPostalCode.cpp.

3.1.3.5 getPrev()

```
BlockPostalCode * BlockPostalCode::getPrev () const
```

Gets the predecessor block pointer.

Returns

A pointer to the previous BlockPostalCode.

Definition at line 68 of file BlockPostalCode.cpp.

References BlockPostalCode().

3.1.3.6 getNext()

```
BlockPostalCode * BlockPostalCode::getNext () const
```

Gets the successor block pointer.

Returns

A pointer to the next BlockPostalCode.

Definition at line 77 of file BlockPostalCode.cpp.

References BlockPostalCode().

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

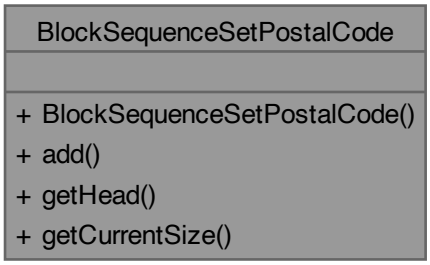
- source/BlockPostalCode.h
- source/BlockPostalCode.cpp

3.2 BlockSequenceSetPostalCode Class Reference

Manages a linked list of BlockPostalCode objects.

```
#include <BlockSequenceSetPostalCode.h>
```

Collaboration diagram for BlockSequenceSetPostalCode:



Public Member Functions

- BlockSequenceSetPostalCode ()
  Default constructor.
- bool add (const HeaderRecordPostalCodeItem &newHeaderPostalCodeItem)
  Adds a new header postal code item as a BlockPostalCode.
- BlockPostalCode getHead () const
  Retrieves the head block by value.
- int getCurrentSize () const
  Gets the number of blocks stored in the sequence.

3.2.1 Detailed Description

Manages a linked list of BlockPostalCode objects.

Each BlockPostalCode contains:

- A HeaderRecordPostalCodeItem
- A predecessor pointer (prev)
- A successor pointer (next)

The BlockSequenceSetPostalCode class stores:

- headBlock → pointer to the first block
- tailBlock → pointer to the last block
- itemCount → number of stored blocks

New blocks are appended to the end (tail), maintaining predecessor and successor relationships.

Definition at line 35 of file BlockSequenceSetPostalCode.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BlockSequenceSetPostalCode()

```
BlockSequenceSetPostalCode::BlockSequenceSetPostalCode ()
```

Default constructor.

Creates an empty block sequence set.

Creates an empty block sequence set.

The head and tail block pointers are initialized to nullptr and the item count is set to zero.

Definition at line 19 of file BlockSequenceSetPostalCode.cpp.

3.2.3 Member Function Documentation

3.2.3.1 add()

```
bool BlockSequenceSetPostalCode::add (
            const HeaderRecordPostalCodeItem & newHeaderPostalCodeItem)
```

Adds a new header postal code item as a BlockPostalCode.

Adds a new header postal code item to the end of the sequence set.

Parameters

newHeaderPostalCodeItem	The item to insert into a new block.

Returns

true if the block was successfully created and linked.

The method allocates a new BlockPostalCode, stores the given header item, and then links the new block into the doubly linked list that represents the block sequence set. The predecessor and successor links of the tail and new block are updated to maintain the structure.

Parameters

newHeaderPostalCodeItem	The header record to add as a new block.

Returns

true if the block was successfully added.

Definition at line 51 of file BlockSequenceSetPostalCode.cpp.

References BlockPostalCode::setBlockItem(), BlockPostalCode::setNextRBN(), and BlockPostalCode::setPrevRBN().

3.2.3.2 getHead()

```
BlockPostalCode BlockSequenceSetPostalCode::getHead () const
```

Retrieves the head block by value.

Returns the current head block by value.

Returns

A copy of the first BlockPostalCode in the sequence.

A copy of the first BlockPostalCode in the sequence set.

Precondition

The sequence set must not be empty (headBlock != nullptr).

Definition at line 35 of file BlockSequenceSetPostalCode.cpp.

3.2.3.3 getCurrentSize()

```
int BlockSequenceSetPostalCode::getCurrentSize () const
```

Gets the number of blocks stored in the sequence.

Gets the number of items currently stored in the sequence set.

Returns

The total count of blocks.

The total count of HeaderRecordPostalCodeItem objects in the list.

Definition at line 25 of file BlockSequenceSetPostalCode.cpp.

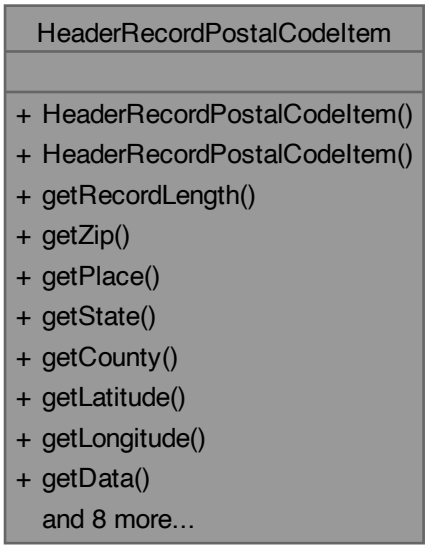
The documentation for this class was generated from the following files:

- source/BlockSequenceSetPostalCode.h
- source/BlockSequenceSetPostalCode.cpp

3.3 HeaderRecordPostalCodeItem Class Reference

```
#include <HeaderRecordPostalCodeItem.h>
```

Collaboration diagram for HeaderRecordPostalCodeItem:



Public Member Functions

- HeaderRecordPostalCodeItem ()

  Default constructor initializing member variables to default values.
- HeaderRecordPostalCodeItem (int r, int z, const string &p, const string &s, const string &c, double lat, double lon)

  Parameterized constructor to initialize a HeaderRecordPostalCodeItem with specific values.
- int getRecordLength () const
- int getZip () const

  Get the ZIP code of the postal code item.
- string getPlace () const

  Get the place name of the postal code item.
- string getState () const

  Get the state name of the postal code item.
- string getCounty () const

  Get the county name of the postal code item.
- double getLatitude () const

  Get the latitude of the postal code item.
- double getLongitude () const

  Get the longitude of the postal code item.
- string getData () const
- void setRecordLength (int newRecordLength)
- void setZip (int newZip)

  Set the ZIP code of the postal code item.
- void setPlace (const string &newPlace)

  Set the place name of the postal code item.
- void setState (const string &newState)

  Set the state name of the postal code item.
- void setCounty (const string &newCounty)

  Set the county name of the postal code item.
- void setLatitude (double newLat)

  Set the latitude of the postal code item.
- void setLongitude (double newLon)

  Set the longitude of the postal code item.
- void printInfo () const

  Print the postal code item's information in a formatted manner.

3.3.1 Detailed Description

Definition at line 13 of file HeaderRecordPostalCodeItem.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 HeaderRecordPostalCodeItem() [1/2]

```
HeaderRecordPostalCodeItem::HeaderRecordPostalCodeItem ()
```

Default constructor initializing member variables to default values.

zip is set to 0, place, state, and county are set to empty strings, and latitude and longitude are set to 0.0. This ensures that a HeaderRecordPostalCodeItem object starts with a known state.

Note

This constructor can be used to create an empty HeaderRecordPostalCodeItem object, which can later be populated with actual data using the setter methods.

See also

HeaderRecordPostalCodeItem(int, const string&, const string&, const string&, double, double)

setZip(int)

setPlace(const string&)

setState(const string&)

setCounty(const string&)

setLatitude(double)

setLongitude(double)

printInfo() const

Definition at line 31 of file HeaderRecordPostalCodeItem.cpp.

3.3.2.2 HeaderRecordPostalCodeItem() [2/2]

```
HeaderRecordPostalCodeItem::HeaderRecordPostalCodeItem (
            int r,
            int z,
            const string & p,
            const string & s,
            const string & c,
            double lat,
            double lon)
```

Parameterized constructor to initialize a HeaderRecordPostalCodeItem with specific values.

Parameters

z	The ZIP code (integer).
p	The place name (string).
s	The state name (string).
c	The county name (string).
lat	The latitude (double).
lon	The longitude (double). This constructor allows for the creation of a fully initialized HeaderRecordPostalCodeItem object.

Note

Ensure that the provided values are valid and meaningful for the postal code entry.

Definition at line 53 of file HeaderRecordPostalCodeItem.cpp.

3.3.3 Member Function Documentation

3.3.3.1 getRecordLength()

```
int HeaderRecordPostalCodeItem::getRecordLength () const
```

Definition at line 64 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.2 getZip()

```
int HeaderRecordPostalCodeItem::getZip () const
```

Get the ZIP code of the postal code item.

Returns

The ZIP code as an integer.

Definition at line 73 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.3 getPlace()

```
string HeaderRecordPostalCodeItem::getPlace () const
```

Get the place name of the postal code item.

Returns

The place name as a string.

Definition at line 82 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.4 getState()

```
string HeaderRecordPostalCodeItem::getState () const
```

Get the state name of the postal code item.

Returns

The state name as a string.

Definition at line 91 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.5 getCounty()

```
string HeaderRecordPostalCodeItem::getCounty () const
```

Get the county name of the postal code item.

Returns

The county name as a string.

Note

County names may vary in format and length depending on the region. Ensure that the county name is correctly formatted for display or processing.

Definition at line 102 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.6 getLatitude()

```
double HeaderRecordPostalCodeItem::getLatitude () const
```

Get the latitude of the postal code item.

Returns

The latitude as a double.

Note

Latitude values are typically in the range of -90 to 90 degrees.

Definition at line 112 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.7 getLongitude()

```
double HeaderRecordPostalCodeItem::getLongitude () const
```

Get the longitude of the postal code item.

Returns

The longitude as a double.

Note

Longitude values are typically in the range of -180 to 180 degrees.

Definition at line 122 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.8 getData()

```
string HeaderRecordPostalCodeItem::getData () const
```

Definition at line 127 of file HeaderRecordPostalCodeItem.cpp.

References getCounty(), getLatitude(), getLongitude(), getPlace(), getState(), and getZip().

3.3.3.9 setRecordLength()

```
void HeaderRecordPostalCodeItem::setRecordLength (
            int newRecordLength)
```

Definition at line 133 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.10 setZip()

```
void HeaderRecordPostalCodeItem::setZip (
            int newZip)
```

Set the ZIP code of the postal code item.

Parameters

newZip	The new ZIP code to be set (integer).

Note

Ensure that the new ZIP code is a valid integer value.

Definition at line 143 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.11 setPlace()

```
void HeaderRecordPostalCodeItem::setPlace (
            const string & newPlace)
```

Set the place name of the postal code item.

Parameters

newPlace	The new place name to be set (string).

Note

Ensure that the new place name is a valid string value.

Definition at line 153 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.12 setState()

```
void HeaderRecordPostalCodeItem::setState (
            const string & newState)
```

Set the state name of the postal code item.

Parameters

newState	The new state name to be set (string).

Note

Ensure that the new state name is a valid string value.

Definition at line 163 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.13 setCounty()

```
void HeaderRecordPostalCodeItem::setCounty (
            const string & newCounty)
```

Set the county name of the postal code item.

Parameters

newCounty	The new county name to be set (string).

Note

Ensure that the new county name is a valid string value.

Definition at line 173 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.14 setLatitude()

```
void HeaderRecordPostalCodeItem::setLatitude (
            double newLat)
```

Set the latitude of the postal code item.

Parameters

newLat	The new latitude to be set (double).

Note

Ensure that the new latitude is within the valid range of -90 to 90 degrees. Invalid latitude values may lead to incorrect geographical representations.

Definition at line 184 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.15 setLongitude()

```
void HeaderRecordPostalCodeItem::setLongitude (
            double newLon)
```

Set the longitude of the postal code item.

Parameters

newLon	The new longitude to be set (double).

Note

Ensure that the new longitude is within the valid range of -180 to 180 degrees. Invalid longitude values may lead to incorrect geographical representations.

Definition at line 195 of file HeaderRecordPostalCodeItem.cpp.

3.3.3.16 printInfo()

```
void HeaderRecordPostalCodeItem::printInfo () const
```

Print the postal code item's information in a formatted manner.

The information includes ZIP code, place name, state, county, latitude, and longitude. The output is aligned in columns for better readability.

Note

This method uses standard output (cout) to display the information.

Definition at line 206 of file HeaderRecordPostalCodeItem.cpp.

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

- source/HeaderRecordPostalCodeItem.h
- source/HeaderRecordPostalCodeItem.cpp

3.4 IndexEntry Struct Reference

Collaboration diagram for IndexEntry:

Public Attributes

- int highestKey
- int rbn

3.4.1 Detailed Description

Definition at line 9 of file search_bss.cpp.

3.4.2 Member Data Documentation

3.4.2.1 highestKey

```
int IndexEntry::highestKey
```

Definition at line 10 of file search_bss.cpp.

3.4.2.2 rbn

```
int IndexEntry::rbn
```

Definition at line 11 of file search_bss.cpp.

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

- source/search_bss.cpp

Chapter 4

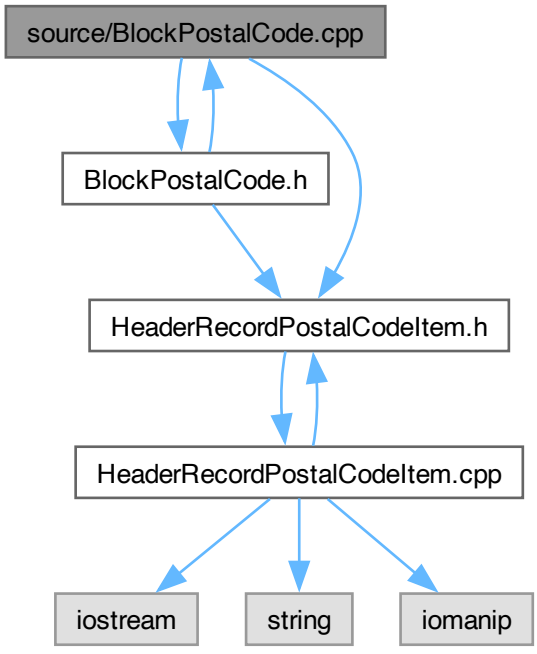
File Documentation

4.1 source/BlockPostalCode.cpp File Reference

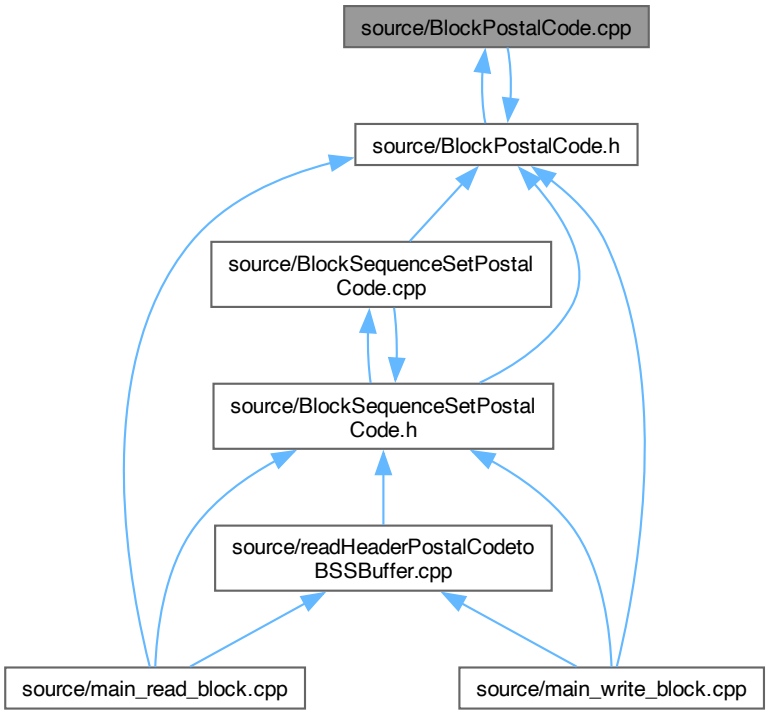
Implements the BlockPostalCode class.

```
#include "BlockPostalCode.h"
#include "HeaderRecordPostalCodeItem.h"
```

Include dependency graph for BlockPostalCode.cpp:



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



4.1.1 Detailed Description

Implements the BlockPostalCode class.

Definition in file BlockPostalCode.cpp.

4.2 BlockPostalCode.cpp

Go to the documentation of this file.

```
00001
00005
00006 #include "BlockPostalCode.h"
00007 #include "HeaderRecordPostalCodeItem.h"
00008
00012 BlockPostalCode::BlockPostalCode() : prevRBN(nullptr), nextRBN(nullptr) {}
00013
00018 BlockPostalCode::BlockPostalCode(const HeaderRecordPostalCodeItem &item) : data(item),
      prevRBN(nullptr), nextRBN(nullptr) {}
00019
00026 BlockPostalCode::BlockPostalCode(const HeaderRecordPostalCodeItem &item, BlockPostalCode *prevBlock,
      BlockPostalCode *nextBlock) : data(item), prevRBN(prevBlock), nextRBN(nextBlock) {}
00027
00032 void BlockPostalCode::setBlockItem(const HeaderRecordPostalCodeItem &item)
00033 {
00034     data = item;
00035 }
00036
00041 void BlockPostalCode::setPrevRBN(BlockPostalCode *prevBlock)
00042 {
```

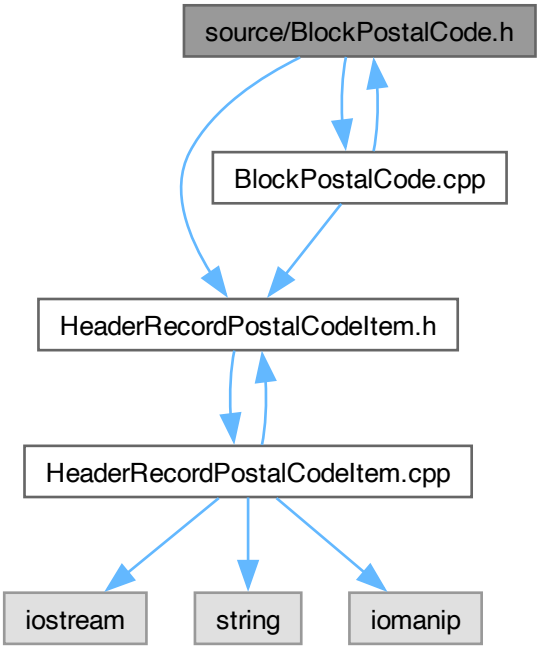
```
00043     prevRBN = prevBlock;
00044 }
00045
00050 void BlockPostalCode::setNextRBN(BlockPostalCode *nextBlock)
00051 {
00052     nextRBN = nextBlock;
00053 }
00054
00059 HeaderRecordPostalCodeItem BlockPostalCode::getBlockItem() const
00060 {
00061     return data;
00062 }
00063
00068 BlockPostalCode *BlockPostalCode::getPrev() const
00069 {
00070     return prevRBN;
00071 }
00072
00077 BlockPostalCode *BlockPostalCode::getNext() const
00078 {
00079     return nextRBN;
00080 }
```

4.3 source/BlockPostalCode.h File Reference

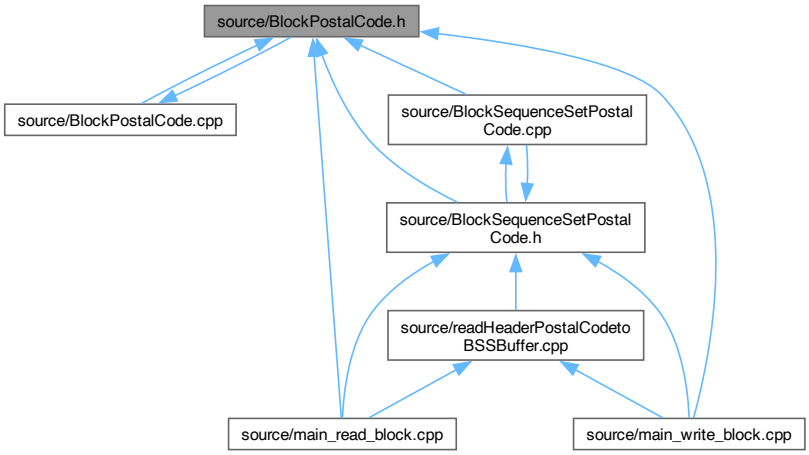
Declares the BlockPostalCode class used in the postal-code block sequence set.

```
#include "HeaderRecordPostalCodeItem.h"
#include "BlockPostalCode.cpp"
```

Include dependency graph for BlockPostalCode.h:



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



Classes

- class BlockPostalCode

  Represents one block that stores a HeaderRecordPostalCodeItem.

4.3.1 Detailed Description

Declares the BlockPostalCode class used in the postal-code block sequence set.

This block stores a single postal-code header record and contains predecessor and successor links (Relative Block Number style) using raw pointers to represent the previous and next block.

Definition in file BlockPostalCode.h.

4.4 BlockPostalCode.h

Go to the documentation of this file.

```
00001 #ifndef BLOCK_POSTAL_CODE
00002 #define BLOCK_POSTAL_CODE
00003
00012
00013 #include "HeaderRecordPostalCodeItem.h"
00014
00026
00027 class BlockPostalCode
00028 {
00029 private:
00031     HeaderRecordPostalCodeItem data;
00032
00035     BlockPostalCode *prevRBN;
00036
00039     BlockPostalCode *nextRBN;
00040
00041 public:
00045     BlockPostalCode();
00046
00051     BlockPostalCode(const HeaderRecordPostalCodeItem &item);
```

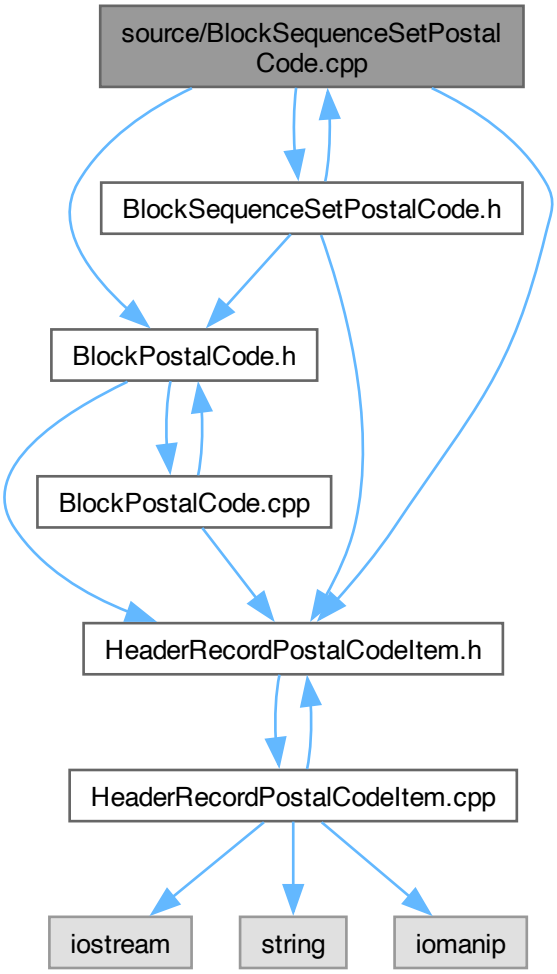
```
00052
00059     BlockPostalCode(const HeaderRecordPostalCodeItem &item, BlockPostalCode *prevBlock,
      BlockPostalCode *nextBlock);
00060
00065     void setBlockItem(const HeaderRecordPostalCodeItem &item);
00066
00071     void setPrevRBN(BlockPostalCode *prevBlock);
00072
00077     void setNextRBN(BlockPostalCode *nextBlock);
00078
00083     HeaderRecordPostalCodeItem getBlockItem() const;
00084
00089     BlockPostalCode *getPrev() const;
00090
00095     BlockPostalCode *getNext() const;
00096 };
00097
00098 #include "BlockPostalCode.cpp"
00099 #endif
```

4.5 source/BlockSequenceSetPostalCode.cpp File Reference

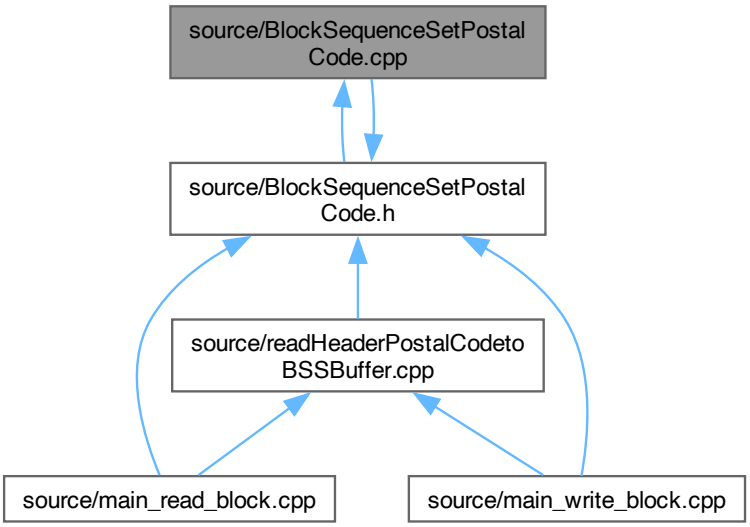
Implements the BlockSequenceSetPostalCode class.

```
#include "BlockSequenceSetPostalCode.h"
#include "BlockPostalCode.h"
#include "HeaderRecordPostalCodeItem.h"
```

Include dependency graph for BlockSequenceSetPostalCode.cpp:



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



4.5.1 Detailed Description

Implements the BlockSequenceSetPostalCode class.

This class manages a sequence of BlockPostalCode nodes that together form the block sequence set structure used for storing header postal code items.

Definition in file BlockSequenceSetPostalCode.cpp.

4.6 BlockSequenceSetPostalCode.cpp

Go to the documentation of this file.

```
00001 #include "BlockSequenceSetPostalCode.h"
00002 #include "BlockPostalCode.h"
00003 #include "HeaderRecordPostalCodeItem.h"
00004
00012
00019 BlockSequenceSetPostalCode::BlockSequenceSetPostalCode() : headBlock(nullptr), itemCount(0) {}
00020
00025 int BlockSequenceSetPostalCode::getCurrentSize() const
00026 {
00027     return itemCount;
00028 }
00029
00035 BlockPostalCode BlockSequenceSetPostalCode::getHead() const
00036 {
00037     return *headBlock;
00038 }
00039
00051 bool BlockSequenceSetPostalCode::add(const HeaderRecordPostalCodeItem &newHeaderPostalCodeItem)
00052 {
00053     BlockPostalCode *newBlock = new BlockPostalCode();
00054     newBlock->setBlockItem(newHeaderPostalCodeItem);
00055
00056     if (headBlock == nullptr || tailBlock == nullptr)
```

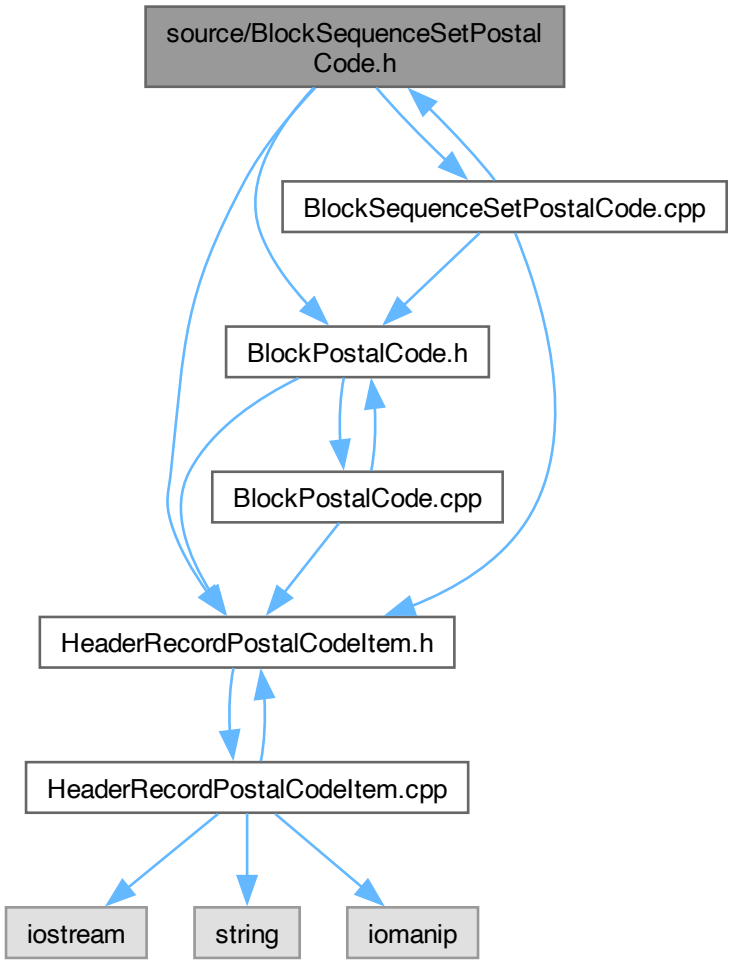
```
00057     {
00058         headBlock = newBlock;
00059         tailBlock = newBlock;
00060     }
00061     else if (headBlock == tailBlock)
00062     {
00063         tailBlock = newBlock;
00064         headBlock->setNextRBN(tailBlock);
00065         tailBlock->setPrevRBN(headBlock);
00066     }
00067     else
00068     {
00069         newBlock->setPrevRBN(tailBlock);
00070         tailBlock->setNextRBN(newBlock);
00071         tailBlock = newBlock;
00072     }
00073
00074     itemCount++;
00075
00076     return true;
00077 }
```

4.7 source/BlockSequenceSetPostalCode.h File Reference

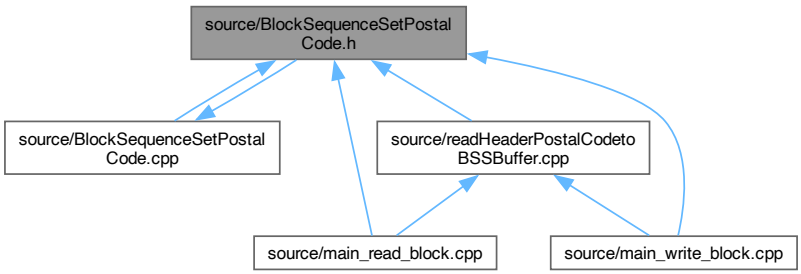
Declares the BlockSequenceSetPostalCode class which manages a doubly linked sequence of BlockPostalCode nodes.

```
#include "BlockPostalCode.h"
#include "HeaderRecordPostalCodeItem.h"
#include "BlockSequenceSetPostalCode.cpp"
```

Include dependency graph for BlockSequenceSetPostalCode.h:



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



Classes

- class BlockSequenceSetPostalCode

  Manages a linked list of BlockPostalCode objects.

4.7.1 Detailed Description

Declares the BlockSequenceSetPostalCode class which manages a doubly linked sequence of BlockPostalCode nodes.

This class forms the Block Sequence Set (BSS) structure used to store postal header records in a linked-block format. Each block is connected using predecessor/successor links similar to Relative Block Number (RBN) behavior.

Definition in file BlockSequenceSetPostalCode.h.

4.8 BlockSequenceSetPostalCode.h

Go to the documentation of this file.

```
00001 #ifndef BLOCK_SEQUENCE_SET_POSTAL_CODE
00002 #define BLOCK_SEQUENCE_SET_POSTAL_CODE
00003
00013
00014 #include "BlockPostalCode.h"
00015 #include "HeaderRecordPostalCodeItem.h"
00016
00035 class BlockSequenceSetPostalCode
00036 {
00037 private:
00038     BlockPostalCode *headBlock;
00039     BlockPostalCode *tailBlock;
00040     int itemCount;
00041
00042 public:
00046     BlockSequenceSetPostalCode();
00047
00053     bool add(const HeaderRecordPostalCodeItem &newHeaderPostalCodeItem);
00054
00059     BlockPostalCode getHead() const;
00060
00065     int getCurrentSize() const;
00066 };
00067
00068 #include "BlockSequenceSetPostalCode.cpp"
00069 #endif
```

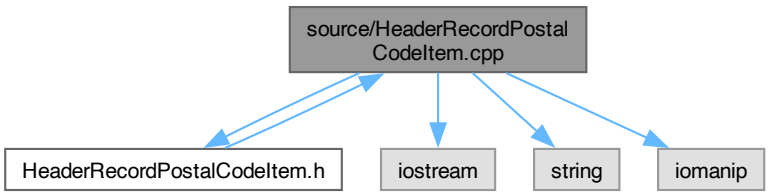
4.9 source/HeaderRecordPostalCodeItem.cpp File Reference

Defines and manages individual HeaderRecord postal code items.

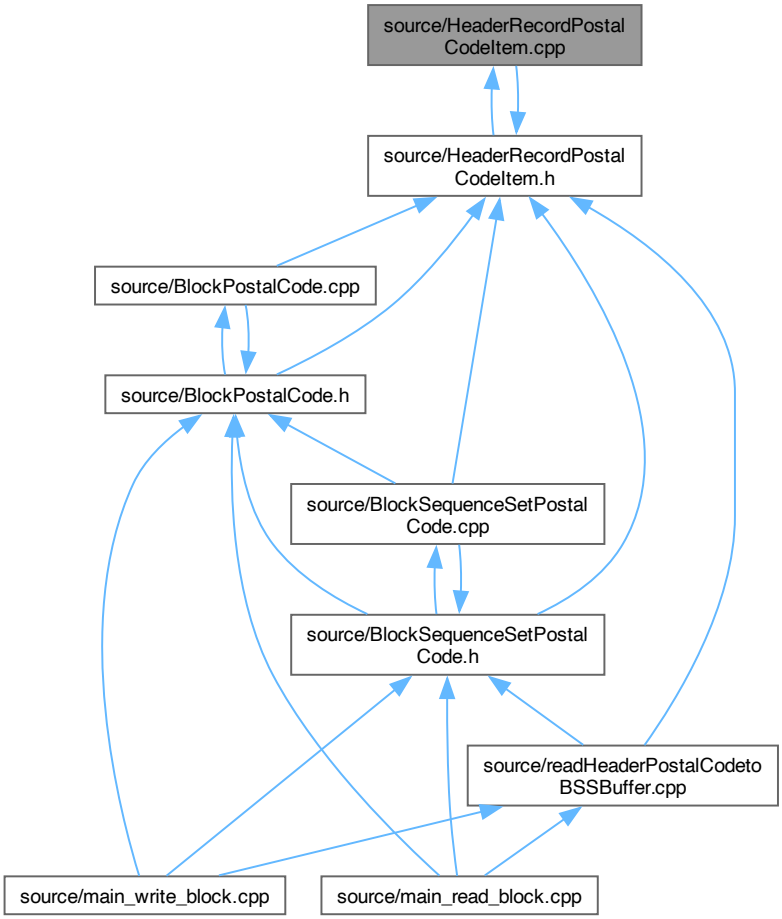
```
#include "HeaderRecordPostalCodeItem.h"
#include <iostream>
#include <string>
```

```
#include <iomanip>
```

Include dependency graph for HeaderRecordPostalCodeItem.cpp:



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



4.9.1 Detailed Description

Defines and manages individual HeaderRecord postal code items.

Author

Mahad Farah, Kariniemi Carson, Tran Minh Quan, Rogers Mitchell, Asfaw Abel

Date

2025-10-17

Definition in file HeaderRecordPostalCodeItem.cpp.

4.10 HeaderRecordPostalCodeItem.cpp

Go to the documentation of this file.

```
00001
00007
00008 #include "HeaderRecordPostalCodeItem.h"
00009 #include <iostream>
00010 #include <string>
00011 #include <iomanip>
00012
00013 using namespace std;
00014
00031 HeaderRecordPostalCodeItem::HeaderRecordPostalCodeItem()
00032 {
00033     recordLength = 0;
00034     zip = 0;
00035     place = "";
00036     state = "";
00037     county = "";
00038     latitude = 0;
00039     longitude = 0;
00040 }
00041
00053 HeaderRecordPostalCodeItem::HeaderRecordPostalCodeItem(int r, int z, const string &p, const string &s,
      const string &c, double lat, double lon)
00054 {
00055     recordLength = r;
00056     zip = z;
00057     place = p;
00058     state = s;
00059     county = c;
00060     latitude = lat;
00061     longitude = lon;
00062 }
00063
00064 int HeaderRecordPostalCodeItem::getRecordLength() const
00065 {
00066     return recordLength;
00067 }
00068
00073 int HeaderRecordPostalCodeItem::getZip() const
00074 {
00075     return zip;
00076 }
00077
00082 string HeaderRecordPostalCodeItem::getPlace() const
00083 {
00084     return place;
00085 }
00086
00091 string HeaderRecordPostalCodeItem::getState() const
00092 {
00093     return state;
00094 }
00095
00102 string HeaderRecordPostalCodeItem::getCounty() const
00103 {
00104     return county;
00105 }
00106
00112 double HeaderRecordPostalCodeItem::getLatitude() const
00113 {
00114     return latitude;
00115 }
00116
00122 double HeaderRecordPostalCodeItem::getLongitude() const
```

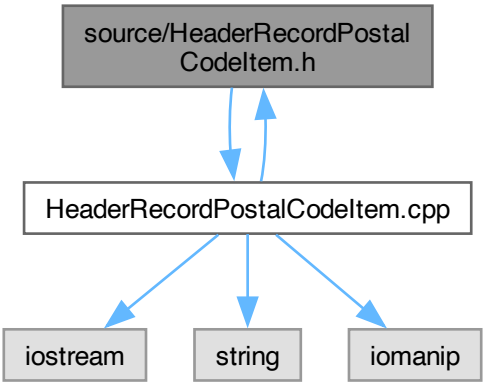
```
00123 {
00124     return longitude;
00125 }
00126
00127 string HeaderRecordPostalCodeItem::getData() const
00128 {
00129     string zipCodeData = to_string(getZip()) + "," + getPlace() + "," + getState() + "," + getCounty()
      + "," + to_string(getLatitude()) + "," + to_string(getLongitude());
00130     return zipCodeData;
00131 }
00132
00133 void HeaderRecordPostalCodeItem::setRecordLength(int newRecordLength)
00134 {
00135     recordLength = newRecordLength;
00136 }
00137
00143 void HeaderRecordPostalCodeItem::setZip(int newZip)
00144 {
00145     zip = newZip;
00146 }
00147
00153 void HeaderRecordPostalCodeItem::setPlace(const string &newPlace)
00154 {
00155     place = newPlace;
00156 }
00157
00163 void HeaderRecordPostalCodeItem::setState(const string &newState)
00164 {
00165     state = newState;
00166 }
00167
00173 void HeaderRecordPostalCodeItem::setCounty(const string &newCounty)
00174 {
00175     county = newCounty;
00176 }
00177
00184 void HeaderRecordPostalCodeItem::setLatitude(double newLat)
00185 {
00186     latitude = newLat;
00187 }
00188
00195 void HeaderRecordPostalCodeItem::setLongitude(double newLon)
00196 {
00197     longitude = newLon;
00198 }
00199
00206 void HeaderRecordPostalCodeItem::printInfo() const
00207 {
00208     cout « left « setw(5) « recordLength
00209          « setw(10) « zip
00210          « setw(20) « place
00211          « setw(10) « state
00212          « setw(30) « county
00213          « setw(12) « latitude
00214          « setw(12) « longitude
00215          « endl;
00216 }
```

4.11 source/HeaderRecordPostalCodeItem.h File Reference

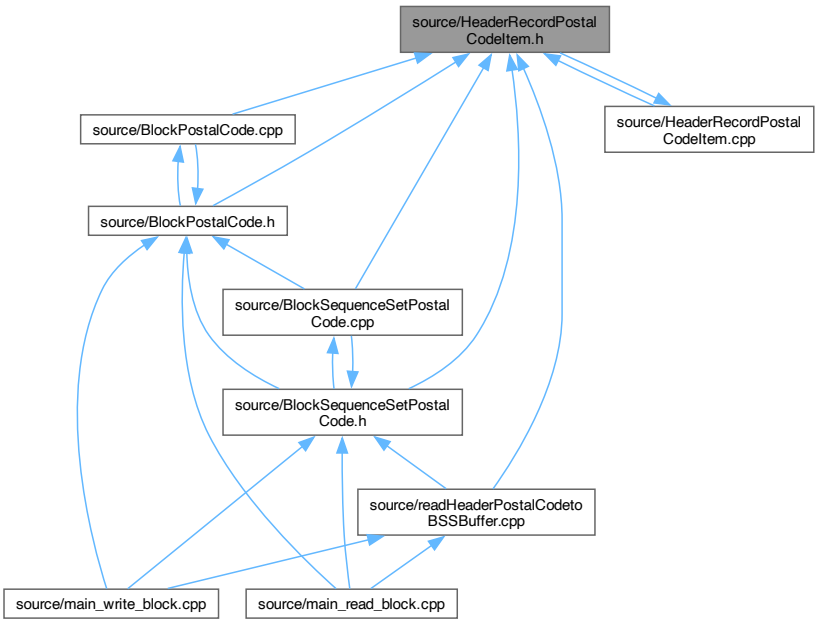
Defines and manages individual HeaderRecord postal code items.

```
#include "HeaderRecordPostalCodeItem.cpp"
```

Include dependency graph for HeaderRecordPostalCodeItem.h:



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



Classes

- class HeaderRecordPostalCodeItem

4.11.1 Detailed Description

Defines and manages individual HeaderRecord postal code items.

Author

Mahad Farah, Kariniemi Carson, Tran Minh Quan, Rogers Mitchell, Asfaw Abel

Date

2025-10-17

Definition in file HeaderRecordPostalCodeItem.h.

4.12 HeaderRecordPostalCodeItem.h

Go to the documentation of this file.

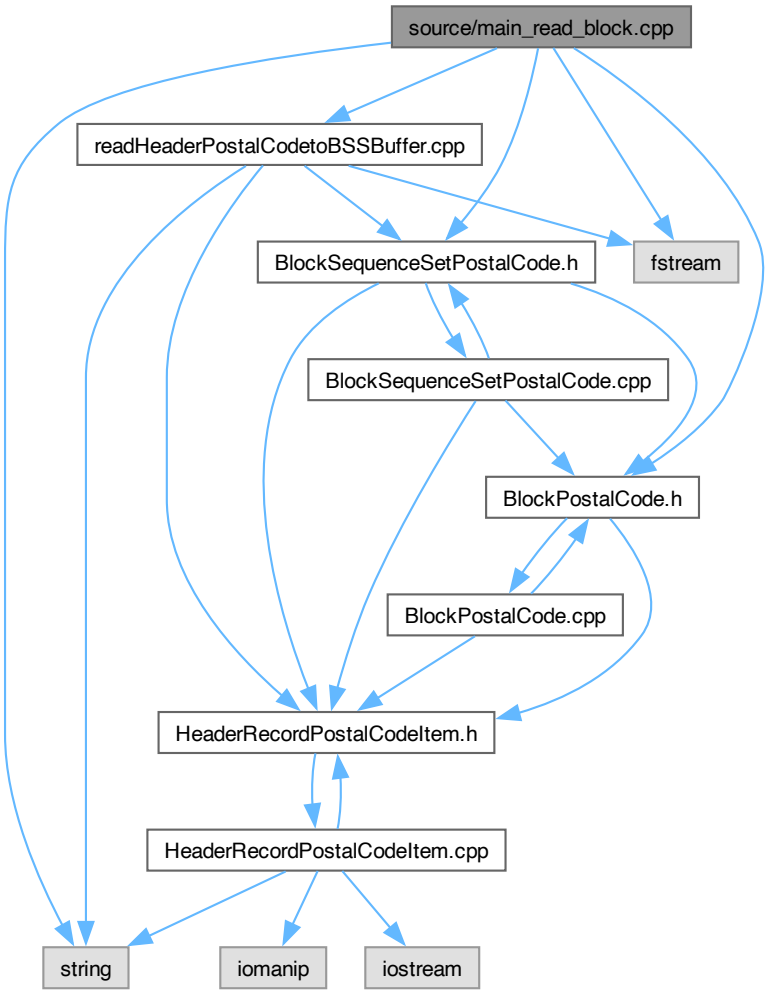
```
00001
00007
00008 #ifndef HEADER_RECORD_POSTAL_CODE_ITEM
00009 #define HEADER_RECORD_POSTAL_CODE_ITEM
00010
00011 using namespace std;
00012
00013 class HeaderRecordPostalCodeItem
00014 {
00015 private:
00016     int recordLength;
00017     int zip;
00018     string place;
00019     string state;
00020     string county;
00021     double latitude;
00022     double longitude;
00023
00024 public:
00041     HeaderRecordPostalCodeItem();
00042
00054     HeaderRecordPostalCodeItem(int r, int z, const string &p, const string &s, const string &c, double
      lat, double lon);
00055
00056     int getRecordLength() const;
00057
00062     int getZip() const;
00063
00068     string getPlace() const;
00069
00074     string getState() const;
00075
00082     string getCounty() const;
00083
00089     double getLatitude() const;
00090
00096     double getLongitude() const;
00097
00098     string getData() const;
00099
00100     void setRecordLength(int newRecordLength);
00101
00107     void setZip(int newZip);
00108
00114     void setPlace(const string &newPlace);
00115
00121     void setState(const string &newState);
00122
00128     void setCounty(const string &newCounty);
00129
00136     void setLatitude(double newLat);
00137
00144     void setLongitude(double newLon);
00145
00152     void printInfo() const;
00153 };
00154
00155 #include "HeaderRecordPostalCodeItem.cpp"
00156 #endif
```

4.13 source/main_read_block.cpp File Reference

Reads a blocked sequence set (BSS) of postal codes and prints block records.

```
#include <string>
#include "BlockPostalCode.h"
#include "BlockSequenceSetPostalCode.h"
#include "readHeaderPostalCodetoBSSBuffer.cpp"
#include <fstream>
```

Include dependency graph for main_read_block.cpp:



Functions

- int main ()

  Program entry point.

4.13.1 Detailed Description

Reads a blocked sequence set (BSS) of postal codes and prints block records.

Definition in file main_read_block.cpp.

4.13.2 Function Documentation

4.13.2.1 main()

```
int main ()
```

Program entry point.

Loads postal code data into a Block Sequence Set and prints each block.

The output format for each block is:
`recordLength data prevZip nextZip`

where "NULL" is used when a link does not exist.

Returns

int Exit status

< Input file containing header + length-indicated records

< The full Block Sequence Set structure

< Populate BSS from data file

< Pointer to first block

< Move to next block

< Advance to next block

Definition at line 26 of file main_read_block.cpp.

References BlockPostalCode::getBlockItem(), HeaderRecordPostalCodeItem::getData(), BlockSequenceSetPostalCode::getHead(), BlockPostalCode::getNext(), BlockPostalCode::getPrev(), HeaderRecordPostalCodeItem::getRecordLength(), HeaderRecordPostalCodeItem::getZip(), and inputDatatoBlockSequenceSet().

4.14 main_read_block.cpp

Go to the documentation of this file.

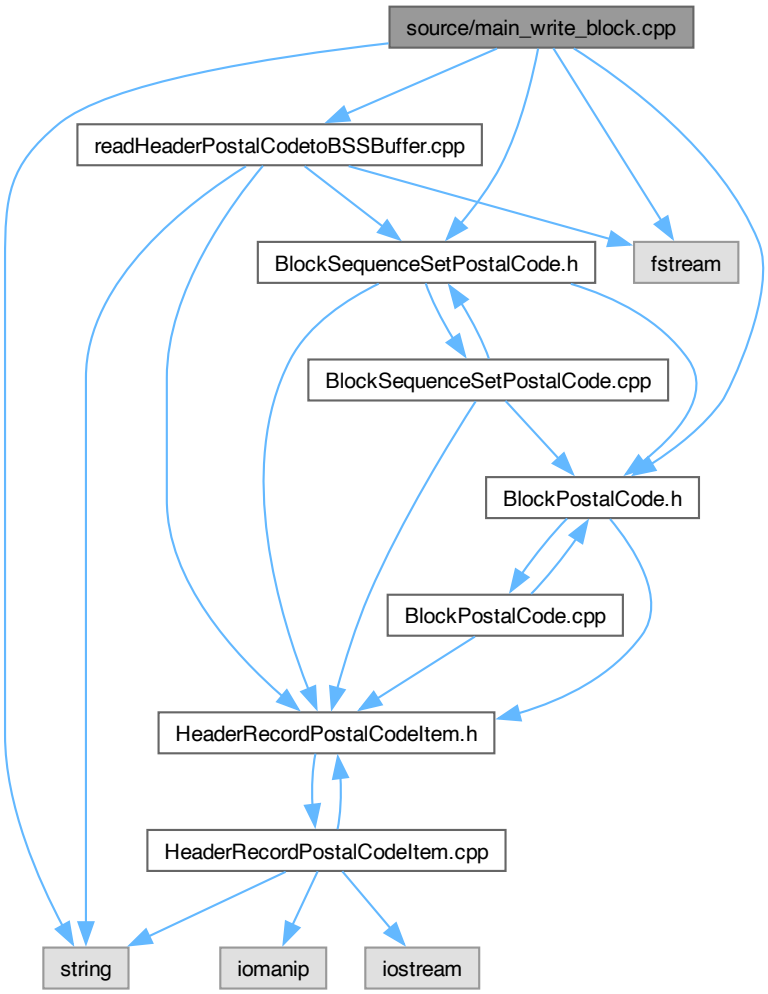
```
00001
00005
00006 #include <string>
00007 #include "BlockPostalCode.h"
00008 #include "BlockSequenceSetPostalCode.h"
00009 #include "readHeaderPostalCodetoBSSBuffer.cpp"
00010 #include <fstream>
00011
00012 using namespace std;
00013
00026 int main()
00027 {
00028     string fileName = "us_postal_codes_length_indicated_header_record.txt";
00029
00030     BlockSequenceSetPostalCode myBlockSequenceSetPostalCode;
00031
00032     inputDatatoBlockSequenceSet(myBlockSequenceSetPostalCode, fileName);
00033
00034     BlockPostalCode myBlock = myBlockSequenceSetPostalCode.getHead();
00035
00036     // Write first block
00037     string blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00038                          myBlock.getBlockItem().getData() + " NULL " +
00039                          to_string(myBlock.getNext()->getBlockItem().getZip());
00040
00041     cout « blockRecord « endl;
00042
00043     myBlock = *myBlock.getNext();
00044
00045     // Loop through middle blocks
00046     while (myBlock.getNext() != nullptr)
00047     {
00048         blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00049                       myBlock.getBlockItem().getData() + " " +
00050                       to_string(myBlock.getPrev()->getBlockItem().getZip()) + " " +
00051                       to_string(myBlock.getNext()->getBlockItem().getZip());
00052
00053         cout « blockRecord « endl;
00054
00055         myBlock = *myBlock.getNext();
00056     }
00057
00058     // Write last block
00059     blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00060                   myBlock.getBlockItem().getData() + " " +
00061                   to_string(myBlock.getPrev()->getBlockItem().getZip()) + " NULL";
00062
00063     cout « blockRecord « endl;
00064
00065     return 0;
00066 }
```

4.15 source/main_write_block.cpp File Reference

Writes a blocked sequence set (BSS) of postal codes to a file.

```
#include <string>
#include "BlockPostalCode.h"
#include "BlockSequenceSetPostalCode.h"
#include "readHeaderPostalCodetoBSSBuffer.cpp"
#include <fstream>
```

Include dependency graph for main_write_block.cpp:



Functions

- int main ()

  Program entry point.

4.15.1 Detailed Description

Writes a blocked sequence set (BSS) of postal codes to a file.

Output format:

```
HeaderRecord Data PrevZip NextZip
```

Definition in file main_write_block.cpp.

4.15.2 Function Documentation

4.15.2.1 main()

```
int main ()
```

Program entry point.

Loads postal code data into a Block Sequence Set and writes block data to block_sequence_set_data.txt.

Block records include record length, data, previous block ZIP, and next block ZIP. "NULL" is written where a link does not exist.

Returns

int Exit status

< Input file for BSS population

< BSS object storing all blocks

< Fill BSS from input file

< Output file for block sequence set

< First block

< Move to next block

< Advance to next block

Definition at line 29 of file main_write_block.cpp.

References BlockPostalCode::getBlockItem(), HeaderRecordPostalCodeItem::getData(), BlockSequenceSetPostalCode::getHead(), BlockPostalCode::getNext(), BlockPostalCode::getPrev(), HeaderRecordPostalCodeItem::getRecordLength(), HeaderRecordPostalCodeItem::getZip(), and inputDatatoBlockSequenceSet().

4.16 main_write_block.cpp

Go to the documentation of this file.

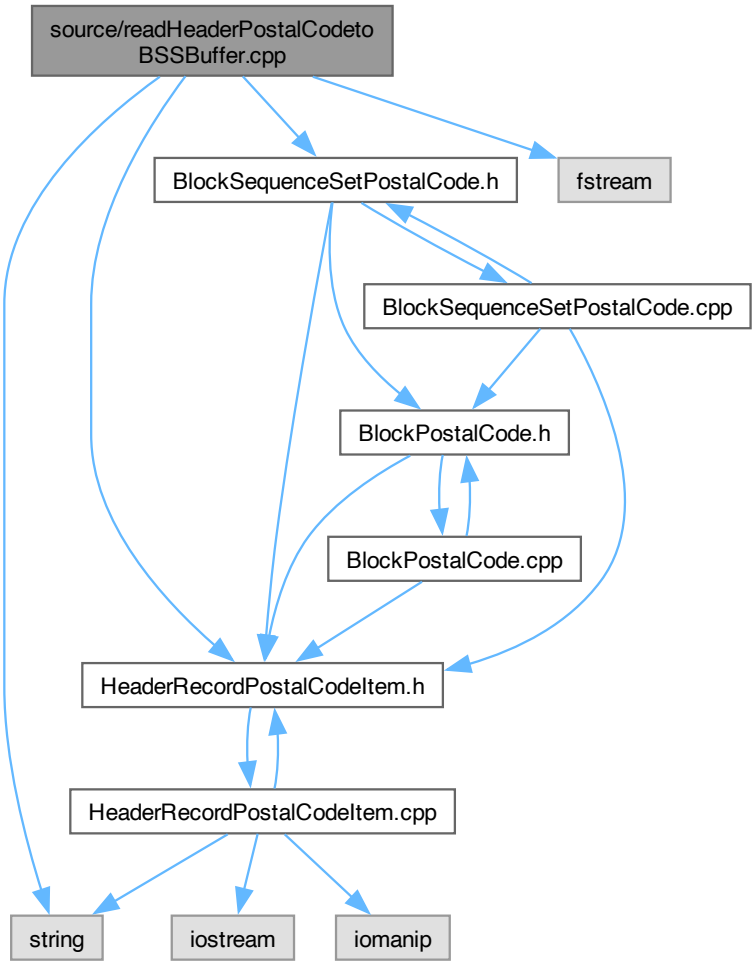
```
00001
00010
00011 #include <string>
00012 #include "BlockPostalCode.h"
00013 #include "BlockSequenceSetPostalCode.h"
00014 #include "readHeaderPostalCodetoBSSBuffer.cpp"
00015 #include <fstream>
00016
00017 using namespace std;
00018
00029 int main()
00030 {
00031     string fileName = "us_postal_codes_length_indicated_header_record.txt";
00032
00033     BlockSequenceSetPostalCode myBlockSequenceSetPostalCode;
00034
00035     inputDatatoBlockSequenceSet(myBlockSequenceSetPostalCode, fileName);
00036
00037     ofstream outputFile("block_sequence_set_data.txt");
00038
00039     BlockPostalCode myBlock = myBlockSequenceSetPostalCode.getHead();
```

```
00040
00041     // Write first block
00042     string blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00043                          myBlock.getBlockItem().getData() + " NULL " +
00044                          to_string(myBlock.getNext()->getBlockItem().getZip());
00045
00046     outputFile « blockRecord « endl;
00047
00048     myBlock = *myBlock.getNext();
00049
00050     // Loop through middle blocks
00051     while (myBlock.getNext() != nullptr)
00052     {
00053         blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00054                       myBlock.getBlockItem().getData() + " " +
00055                       to_string(myBlock.getPrev()->getBlockItem().getZip()) + " " +
00056                       to_string(myBlock.getNext()->getBlockItem().getZip());
00057
00058         outputFile « blockRecord « endl;
00059
00060         myBlock = *myBlock.getNext();
00061     }
00062
00063     // Write last block
00064     blockRecord = to_string(myBlock.getBlockItem().getRecordLength()) + " " +
00065                   myBlock.getBlockItem().getData() + " " +
00066                   to_string(myBlock.getPrev()->getBlockItem().getZip()) + " NULL";
00067
00068     outputFile « blockRecord « endl;
00069
00070     outputFile.close();
00071
00072     return 0;
00073 }
```

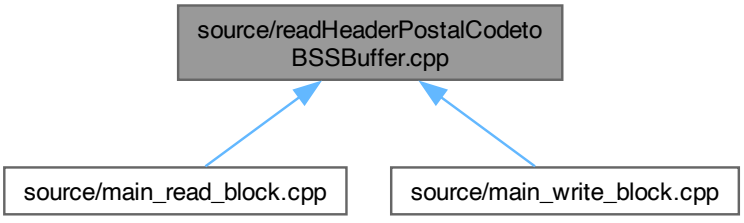
4.17 source/readHeaderPostalCodetoBSSBuffer.cpp File Reference

```
#include <string>
#include "HeaderRecordPostalCodeItem.h"
#include "BlockSequenceSetPostalCode.h"
#include <fstream>
```

Include dependency graph for readHeaderPostalCodetoBSSBuffer.cpp:



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



Functions

- void inputDatatoBlockSequenceSet (BlockSequenceSetPostalCode &inputList, string fileName)

4.17.1 Function Documentation

4.17.1.1 inputDatatoBlockSequenceSet()

```
void inputDatatoBlockSequenceSet (
            BlockSequenceSetPostalCode & inputList,
            string fileName)
```

Definition at line 10 of file readHeaderPostalCodetoBSSBuffer.cpp.

References BlockSequenceSetPostalCode::add(), HeaderRecordPostalCodeItem::setCounty(), HeaderRecordPostalCodeItem::setLa
HeaderRecordPostalCodeItem::setLongitude(), HeaderRecordPostalCodeItem::setPlace(), HeaderRecordPostalCodeItem::setRecord
HeaderRecordPostalCodeItem::setState(), and HeaderRecordPostalCodeItem::setZip().

4.18 readHeaderPostalCodetoBSSBuffer.cpp

Go to the documentation of this file.

```
00001 // This is the buffer file to read the header record to the block sequence set
00002
00003 #include <string>
00004 #include "HeaderRecordPostalCodeItem.h"
00005 #include "BlockSequenceSetPostalCode.h"
00006 #include <fstream>
00007
00008 using namespace std;
00009
00010 void inputDatatoBlockSequenceSet(BlockSequenceSetPostalCode &inputList, string fileName)
00011 {
00012     HeaderRecordPostalCodeItem item;
00013     string line = "";
00014     int location = 0;
00015
00016     ifstream myFile;
00017     myFile.open(fileName);
00018
00019     // Skip the header: "zip,place,state,county,latitude,longitude"
00020     getline(myFile, line);
00021
00022     while (getline(myFile, line))
00023     {
00024         // Record Length
00025         item.setRecordLength(stoi(line.substr(0, 2)));
00026         line = line.substr(2, line.length());
00027         // ZIP
00028         location = line.find(",");
00029         item.setZip(stoi(line.substr(0, location)));
00030         line = line.substr(location + 1, line.length());
00031
00032         // Place
00033         location = line.find(",");
00034         item.setPlace(line.substr(0, location));
00035         line = line.substr(location + 1, line.length());
00036
00037         // State
00038         location = line.find(",");
00039         item.setState(line.substr(0, location));
00040         line = line.substr(location + 1, line.length());
00041
00042         // County
00043         location = line.find(",");
00044         item.setCounty(line.substr(0, location));
00045         line = line.substr(location + 1, line.length());
00046
00047         // Latitude
```

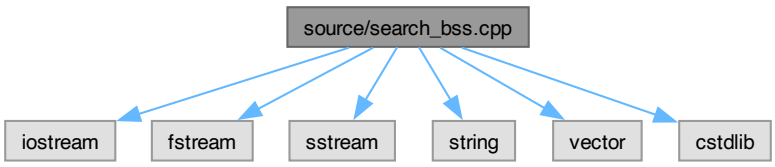
```
00048         location = line.find(",");
00049         item.setLatitude(stod(line.substr(0, location)));
00050         line = line.substr(location + 1, line.length());
00051
00052         // Longitude (last part of the line)
00053         item.setLongitude(stod(line));
00054
00055         // Add it to our list
00056         inputList.add(item);
00057     }
00058
00059     myFile.close();
00060 }
```

4.19 source/README.md File Reference

4.20 source/search_bss.cpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <cstdlib>
```

Include dependency graph for search_bss.cpp:



Classes

- struct IndexEntry

Functions

- int extractZipFromBssLine (const std::string &line)
- bool readBlockLineByRbn (const std::string &dataFile, int rbn, std::string &outLine)
- void printRecordFromBssLine (const std::string &line)
- std::vector< IndexEntry > buildIndexFromDataFile (const std::string &dataFile)
- void writeIndexToFile (const std::string &indexFile, const std::vector< IndexEntry > &index)
- std::vector< IndexEntry > readIndexFromFile (const std::string &indexFile)
- int findBlockRbnForZip (const std::vector< IndexEntry > &index, int zipKey)
- int main (int argc, char *argv[])

4.20.1 Function Documentation

4.20.1.1 extractZipFromBssLine()

```
int extractZipFromBssLine (
            const std::string & line)
```

Definition at line 25 of file search_bss.cpp.

4.20.1.2 readBlockLineByRbn()

```
bool readBlockLineByRbn (
            const std::string & dataFile,
            int rbn,
            std::string & outLine)
```

Definition at line 58 of file search_bss.cpp.

4.20.1.3 printRecordFromBssLine()

```
void printRecordFromBssLine (
            const std::string & line)
```

Definition at line 77 of file search_bss.cpp.

4.20.1.4 buildIndexFromDataFile()

```
std::vector< IndexEntry > buildIndexFromDataFile (
            const std::string & dataFile)
```

Definition at line 93 of file search_bss.cpp.

References extractZipFromBssLine().

4.20.1.5 writeIndexToFile()

```
void writeIndexToFile (
            const std::string & indexFile,
            const std::vector< IndexEntry > & index)
```

Definition at line 120 of file search_bss.cpp.

4.20.1.6 readIndexFromFile()

```
std::vector< IndexEntry > readIndexFromFile (
            const std::string & indexFile)
```

Definition at line 131 of file search_bss.cpp.

4.20.1.7 findBlockRbnForZip()

```
int findBlockRbnForZip (
            const std::vector< IndexEntry > & index,
            int zipKey)
```

Definition at line 148 of file search_bss.cpp.

4.20.1.8 main()

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

Definition at line 167 of file search_bss.cpp.

References buildIndexFromDataFile(), extractZipFromBssLine(), findBlockRbnForZip(), printRecordFromBssLine(), readBlockLineByRbn(), readIndexFromFile(), and writeIndexToFile().

4.21 search_bss.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <sstream>
00004 #include <string>
00005 #include <vector>
00006 #include <cstdlib>
00007
00008 // Simple struct for an index entry: highest key in block, and its RBN
00009 struct IndexEntry {
00010     int highestKey;
00011     int rbn;
00012 };
00013
00014 // ---------- Helpers for parsing BSS lines ----------
00015
00016 // Given a line from block_sequence_set_data.txt, extract the ZIP code.
00017 //
00018 // Format your writer uses (one record per block):
00019 //   recordLength SP zip,place,state,county,lat,lon SP prevLink SP nextLink
00020 //
00021 // We:
00022 //   - read recordLength (everything before first space)
00023 //   - grab the substring between firstSpace+1 and secondLastSpace as the CSV "data"
00024 //   - ZIP is everything before the first comma in that data
00025 int extractZipFromBssLine(const std::string &line) {
00026     // First space (after recordLength)
00027     std::size_t firstSpace = line.find(' ');
00028     if (firstSpace == std::string::npos) {
00029         throw std::runtime_error("Malformed BSS line (no first space): " + line);
00030     }
00031
00032     // Last space (before nextLink)
00033     std::size_t lastSpace = line.rfind(' ');
00034     if (lastSpace == std::string::npos || lastSpace <= firstSpace) {
00035         throw std::runtime_error("Malformed BSS line (no last space): " + line);
00036     }
00037
00038     // Second-to-last space (between data and prevLink)
00039     std::size_t secondLastSpace = line.rfind(' ', lastSpace - 1);
00040     if (secondLastSpace == std::string::npos || secondLastSpace <= firstSpace) {
00041         throw std::runtime_error("Malformed BSS line (no second-last space): " + line);
00042     }
00043
00044     // CSV data substring
00045     std::string data = line.substr(firstSpace + 1, secondLastSpace - firstSpace - 1);
00046
00047     // ZIP is before first comma in data
```

```
00048     std::size_t firstComma = data.find(',');
00049     if (firstComma == std::string::npos) {
00050         throw std::runtime_error("Malformed data section (no comma): " + data);
00051     }
00052
00053     int zip = std::stoi(data.substr(0, firstComma));
00054     return zip;
00055 }
00056
00057 // Read the RBN-th line (0-based) from the BSS data file
00058 bool readBlockLineByRbn(const std::string &dataFile, int rbn, std::string &outLine) {
00059     std::ifstream in(dataFile);
00060     if (!in) {
00061         std::cerr « "Error: cannot open data file '" « dataFile « "'\n";
00062         return false;
00063     }
00064
00065     std::string line;
00066     int current = 0;
00067     while (std::getline(in, line)) {
00068         if (current == rbn) {
00069             outLine = line;
00070             return true;
00071         }
00072         ++current;
00073     }
00074     return false; // RBN out of range
00075 }
00076
00077 void printRecordFromBssLine(const std::string &line) {
00078     std::size_t firstSpace = line.find(' ');
00079     std::size_t lastSpace = line.rfind(' ');
00080     std::size_t secondLastSpace = line.rfind(' ', lastSpace - 1);
00081
00082     if (firstSpace == std::string::npos ||
00083         lastSpace == std::string::npos ||
00084         secondLastSpace == std::string::npos) {
00085         std::cout « line « "\n"; // fallback
00086         return;
00087     }
00088
00089     std::string data = line.substr(firstSpace + 1, secondLastSpace - firstSpace - 1);
00090     std::cout « "  Record: " « data « "\n";
00091 }
00092
00093 std::vector<IndexEntry> buildIndexFromDataFile(const std::string &dataFile) {
00094     std::vector<IndexEntry> index;
00095     std::ifstream in(dataFile);
00096     if (!in) {
00097         throw std::runtime_error("Cannot open data file for index build: " + dataFile);
00098     }
00099
00100     std::string line;
00101     int rbn = 0;
00102     while (std::getline(in, line)) {
00103         if (line.empty()) {
00104             ++rbn;
00105             continue;
00106         }
00107         int zip = extractZipFromBssLine(line);
00108         IndexEntry entry{zip, rbn};
00109         index.push_back(entry);
00110         ++rbn;
00111     }
00112
00113     // Assuming BSS is in ascending zip order, index is already sorted by highestKey.
00114     // If not, you could std::sort by highestKey here.
00115
00116     return index;
00117 }
00118
00119 // Write index to a text file: "highestKey rbn" per line
00120 void writeIndexToFile(const std::string &indexFile, const std::vector<IndexEntry> &index) {
00121     std::ofstream out(indexFile);
00122     if (!out) {
00123         throw std::runtime_error("Cannot open index file for writing: " + indexFile);
00124     }
00125     for (const auto &entry : index) {
00126         out « entry.highestKey « " " « entry.rbn « "\n";
00127     }
00128 }
00129
00130 // Read index back from file into RAM
00131 std::vector<IndexEntry> readIndexFromFile(const std::string &indexFile) {
00132     std::vector<IndexEntry> index;
00133     std::ifstream in(indexFile);
00134     if (!in) {
```

```
00135         throw std::runtime_error("Cannot open index file for reading: " + indexFile);
00136     }
00137
00138     int key, rbn;
00139     while (in » key » rbn) {
00140         index.push_back(IndexEntry{key, rbn});
00141     }
00142     return index;
00143 }
00144
00145 // Find the RBN of the block that should contain zipKey using the index.
00146 // We find the first index entry with highestKey >= zipKey (like lower_bound).
00147 // If none is found, return -1 (definitely not in file).
00148 int findBlockRbnForZip(const std::vector<IndexEntry> &index, int zipKey) {
00149     int low = 0;
00150     int high = static_cast<int>(index.size()) - 1;
00151     int resultRbn = -1;
00152
00153     while (low <= high) {
00154         int mid = (low + high) / 2;
00155         if (index[mid].highestKey >= zipKey) {
00156             resultRbn = index[mid].rbn;
00157             high = mid - 1;
00158         } else {
00159             low = mid + 1;
00160         }
00161     }
00162     return resultRbn;
00163 }
00164
00165 // ---------- Main program ----------
00166
00167 int main(int argc, char *argv[]) {
00168     // Defaults (satisfies #12 by being overridable from the command line)
00169     std::string dataFile  = "block_sequence_set_data.txt";
00170     std::string indexFile = "simple_index.txt";
00171     bool rebuildIndex     = false;
00172     std::vector<int> zipsToSearch;
00173
00174     // Parse command line
00175     for (int i = 1; i < argc; ++i) {
00176         std::string arg = argv[i];
00177
00178         if (arg == "-D" && i + 1 < argc) {
00179             dataFile = argv[++i];
00180         } else if (arg == "-I" && i + 1 < argc) {
00181             indexFile = argv[++i];
00182         } else if (arg == "-buildIndex") {
00183             rebuildIndex = true;
00184         } else if (arg.rfind("-Z", 0) == 0 && arg.size() > 2) {
00185             int zip = std::stoi(arg.substr(2));
00186             zipsToSearch.push_back(zip);
00187         } else {
00188             std::cerr « "Warning: unrecognized argument '" « arg « "'\n";
00189         }
00190     }
00191
00192     if (zipsToSearch.empty()) {
00193         std::cerr « "Usage: " « argv[0]
00194                   « " [-D dataFile] [-I indexFile] [-buildIndex] -Z56301 -Z56302 ...\n";
00195         return 1;
00196     }
00197
00198     try {
00199         // Step 1: maybe build index in RAM and write it to disk
00200         if (rebuildIndex) {
00201             std::cout « "Building index from data file '" « dataFile « "'...\n";
00202             auto index = buildIndexFromDataFile(dataFile);
00203             writeIndexToFile(indexFile, index);
00204             std::cout « "Index written to '" « indexFile « "' (" « index.size()
00205                       « " entries).\n";
00206         }
00207
00208         // Step 2: read index file back into RAM
00209         auto index = readIndexFromFile(indexFile);
00210         if (index.empty()) {
00211             std::cerr « "Error: index file '" « indexFile « "' is empty\n";
00212             return 1;
00213         }
00214
00215         // Step 3: search for each requested Zip
00216         for (int zip : zipsToSearch) {
00217             std::cout « "Searching for ZIP " « zip « "...\n";
00218
00219             int rbn = findBlockRbnForZip(index, zip);
00220             if (rbn == -1) {
00221                 std::cout « "  Not found (ZIP is larger than any highestKey in index).\n";
```

```
00222                 continue;
00223             }
00224
00225             std::string blockLine;
00226             if (!readBlockLineByRbn(dataFile, rbn, blockLine)) {
00227                 std::cout « "  Error: block RBN " « rbn
00228                           « " not found in data file '" « dataFile « "'.\n";
00229                 continue;
00230             }
00231
00232             // In your current design, each block contains one record, so just compare.
00233             int blockZip = extractZipFromBssLine(blockLine);
00234             if (blockZip == zip) {
00235                 std::cout « "  Found in block RBN " « rbn « ":\n";
00236                 printRecordFromBssLine(blockLine);
00237             } else {
00238                 std::cout « "  Not found in block RBN " « rbn
00239                           « " (highestKey there is " « blockZip « ").\n";
00240             }
00241         }
00242
00243     } catch (const std::exception &ex) {
00244         std::cerr « "Fatal error: " « ex.what() « "\n";
00245         return 1;
00246     }
00247
00248     return 0;
00249 }
```