# Project 4 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

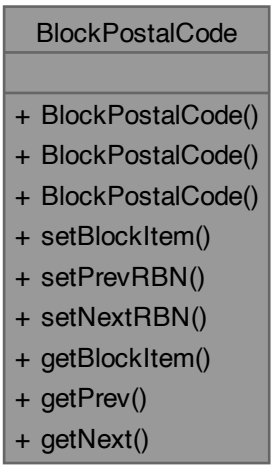
# Class Documentation

## 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:

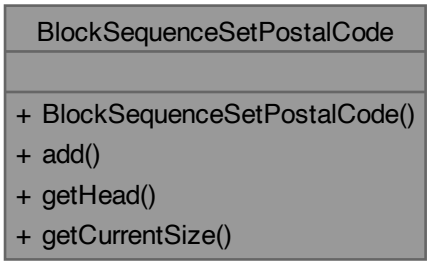
- source/BlockPostalCode.h
- source/BlockPostalCode.cpp

## 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.

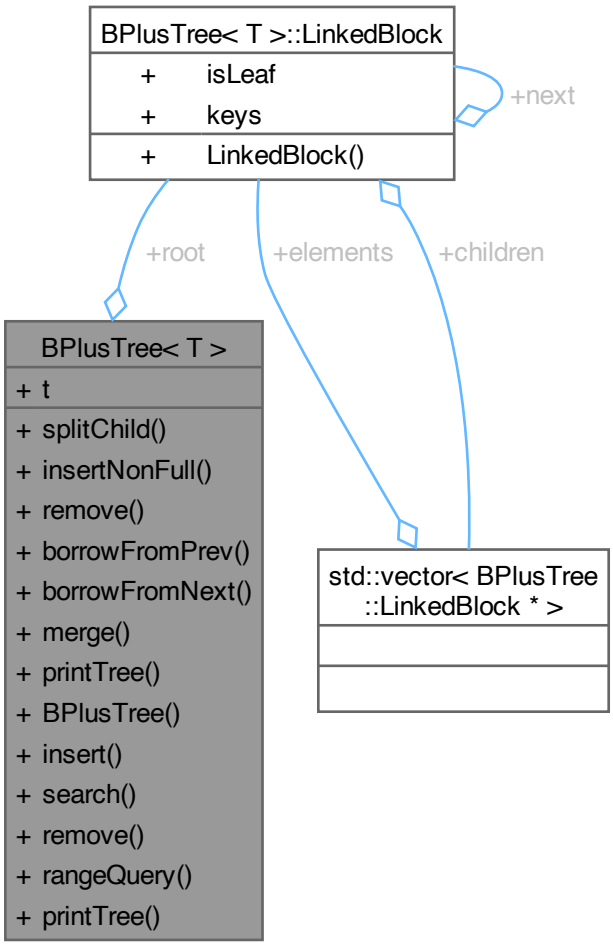
The documentation for this class was generated from the following files:

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

## 3.3 BPlusTree< T > Class Template Reference

B+ tree class template.

Collaboration diagram for BPlusTree< T >:




## Classes

- struct LinkedBlock

  *Node structure representing a B+ tree block.*

## Public Member Functions

- void splitChild (LinkedBlock *parent, int index, LinkedBlock *child)

  *Splits a full child node of an internal node.*
- void insertNonFull (LinkedBlock *linkedBlock, T key)

  *Inserts a key into a non-full node.*
- void remove (LinkedBlock *linkedBlock, T key)

  *Removes a key from a subtree rooted at a given node.*
- void borrowFromPrev (LinkedBlock *linkedBlock, int index)

  *Borrows a key from the previous sibling of a child.*

- void borrowFromNext (LinkedBlock ∗linkedBlock, int index)

  *Borrows a key from the next sibling of a child.*
- void merge (LinkedBlock ∗linkedBlock, int index)

  *Merges a child node with its right sibling.*
- void printTree (LinkedBlock ∗linkedBlock, int level)

  *Prints the keys of the subtree rooted at a given node.*
- BPlusTree (int degree)

  *Constructs a B+ tree with a given minimum degree.*
- void insert (T key)

  *Inserts a key into the B+ tree.*
- bool search (T key)

  *Searches for a key in the B+ tree.*
- void remove (T key)

  *Removes a key from the B+ tree.*
- vector< T > rangeQuery (T lower, T upper)

  *Performs a range query on the B+ tree.*
- void printTree ()

  *Prints the entire B+ tree to standard output.*

**Public Attributes**

- LinkedBlock ∗ root

  *Pointer to the root node of the B+ tree.*
- int t

  *Minimum degree of the B+ tree.*

### 3.3.1 Detailed Description

**template<typename T>**
**class BPlusTree< T >**

B+ tree class template.

Implements a basic B+ tree with insert, remove, search, range query, and printing capabilities. The tree nodes are represented by LinkedBlock structures.

**Template Parameters**

| | |
|---|---|
| *T* | Type of keys stored in the B+ tree. |

Definition at line 17 of file B+tree.cpp.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 BPlusTree()

```
template<typename T>
BPlusTree< T >::BPlusTree (
            int degree)  [inline]
```

Constructs a B+ tree with a given minimum degree.

**Parameters**

| | |
|---|---|
| *degree* | Minimum degree (t) of the B+ tree. |

Definition at line 155 of file B+tree.cpp.

References root, and t.

### 3.3.3 Member Function Documentation

#### 3.3.3.1 splitChild()

```
template<typename T>
void BPlusTree< T >::splitChild (
            LinkedBlock * parent,
            int index,
            LinkedBlock * child)
```

Splits a full child node of an internal node.

Used during insertion when a child node is full. The node is split into two nodes, and a key is promoted into the parent.

**Parameters**

| | |
|---|---|
| *parent* | Pointer to the parent node. |
| *index* | Index of the child in the parent's children vector. |
| *child* | Pointer to the child node to split. |

See BPlusTree::splitChild for detailed description.

Definition at line 210 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, BPlusTree< T >::LinkedBlock::keys, BPlusTree< T >::LinkedBlock::next, and t.

#### 3.3.3.2 insertNonFull()

```
template<typename T>
void BPlusTree< T >::insertNonFull (
            LinkedBlock * linkedBlock,
            T key)
```

Inserts a key into a non-full node.

Inserts a key into a node that is guaranteed to be non-full.

Called by insert() after ensuring that the root is not full.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the node that is guaranteed to be non-full. |
| *key* | Key to insert. |

See BPlusTree::insertNonFull for detailed description.

Definition at line 244 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, insertNonFull(), BPlusTree< T >::LinkedBlock::isLeaf, BPlusTree< T >::LinkedBlock::keys, splitChild(), and t.

#### 3.3.3.3 remove() [1/2]

```
template<typename T>
void BPlusTree< T >::remove (
            LinkedBlock * linkedBlock,
            T key)
```

Removes a key from a subtree rooted at a given node.

Internal recursive remove helper.

Internal recursive helper for the public remove(T key) function.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the current node. |
| *key* | Key to remove. |

See BPlusTree::remove(LinkedBlock*, T) for detailed description.

Definition at line 280 of file B+tree.cpp.

References borrowFromNext(), borrowFromPrev(), BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, BPlusTree< T >::LinkedBlock::keys, merge(), remove(), and t.

#### 3.3.3.4 borrowFromPrev()

```
template<typename T>
void BPlusTree< T >::borrowFromPrev (
            LinkedBlock * linkedBlock,
            int index)
```

Borrows a key from the previous sibling of a child.

Borrows a key from the previous sibling of a child node.

Used during deletion when a child has too few keys and its left sibling can spare a key.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the parent node. |
| *index* | Index of the child in the parent's children vector. |

See BPlusTree::borrowFromPrev for detailed description.

Definition at line 364 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, and BPlusTree< T >::LinkedBlock::keys.

#### 3.3.3.5 borrowFromNext()

```
template<typename T>
void BPlusTree< T >::borrowFromNext (
            LinkedBlock * linkedBlock,
            int index)
```

Borrows a key from the next sibling of a child.

Borrows a key from the next sibling of a child node.

Used during deletion when a child has too few keys and its right sibling can spare a key.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the parent node. |
| *index* | Index of the child in the parent's children vector. |

See BPlusTree::borrowFromNext for detailed description.

Definition at line 389 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, and BPlusTree< T >::LinkedBlock::keys.

#### 3.3.3.6 merge()

```
template<typename T>
void BPlusTree< T >::merge (
            LinkedBlock * linkedBlock,
            int index)
```

Merges a child node with its right sibling.

Used during deletion when both a child and its sibling have the minimum number of keys, and they are combined into a single node.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the parent node. |
| *index* | Index of the left child to merge with its right sibling. |

See BPlusTree::merge for detailed description.

Definition at line 413 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, and BPlusTree< T >::LinkedBlock::keys.

#### 3.3.3.7 printTree() [1/2]

```
template<typename T>
void BPlusTree< T >::printTree (
            LinkedBlock * linkedBlock,
            int level)
```

Prints the keys of the subtree rooted at a given node.

Recursively prints the subtree rooted at a given node.

Recursive helper for the public printTree() function.

**Parameters**

| | |
|---|---|
| *linkedBlock* | Pointer to the current node. |
| *level* | Current depth level in the tree (for indentation). |

See BPlusTree::printTree(LinkedBlock*, int) for detailed description.

Definition at line 442 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::keys, and printTree().

#### 3.3.3.8 insert()

```
template<typename T>
void BPlusTree< T >::insert (
            T key)
```

Inserts a key into the B+ tree.

Inserts a key into the B+ tree, handling root splitting if necessary.

If the root is full, it is split and the tree height increases.

**Parameters**

| | |
|---|---|
| *key* | Key to insert. |

See BPlusTree::insert for detailed description.

Definition at line 550 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, insertNonFull(), root, splitChild(), and t.

#### 3.3.3.9 search()

```
template<typename T>
bool BPlusTree< T >::search (
            T key)
```

Searches for a key in the B+ tree.

Searches for a key starting from the root node.

**Parameters**

| | |
|---|---|
| *key* | Key to search for. |

**Returns**

true If the key is found.

false If the key is not found.

See BPlusTree::search for detailed description.

Definition at line 479 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, BPlusTree< T >::LinkedBlock::keys, and root.

#### 3.3.3.10 remove() [2/2]

```
template<typename T>
void BPlusTree< T >::remove (
            T key)
```

Removes a key from the B+ tree.

Removes a key from the B+ tree, adjusting the root if needed.

Handles root adjustment if it becomes empty after deletion.

**Parameters**

| | |
|---|---|
| *key* | Key to remove. |

See BPlusTree::remove(T) for detailed description.

Definition at line 577 of file B+tree.cpp.

References remove(), and root.

#### 3.3.3.11 rangeQuery()

```
template<typename T>
vector< T > BPlusTree< T >::rangeQuery (
            T lower,
            T upper)
```

Performs a range query on the B+ tree.

Executes a range query on the B+ tree.

Returns all keys in the closed interval [lower, upper].

**Parameters**

| | |
|---|---|
| *lower* | Lower bound of the range (inclusive). |
| *upper* | Upper bound of the range (inclusive). |

**Returns**

vector<T> Collection of keys in the specified range.

Starts from the leaf node where the lower bound would be located and traverses forward using leaf links.

See BPlusTree::rangeQuery for detailed description.

Definition at line 512 of file B+tree.cpp.

References BPlusTree< T >::LinkedBlock::children, BPlusTree< T >::LinkedBlock::isLeaf, BPlusTree< T >::LinkedBlock::keys, BPlusTree< T >::LinkedBlock::next, and root.

#### 3.3.3.12 printTree() [2/2]

```
template<typename T>
void BPlusTree< T >::printTree ()
```

Prints the entire B+ tree to standard output.

Prints the entire B+ tree starting from the root.

Wrapper around the recursive printTree(LinkedBlock*, int) function.

Definition at line 467 of file B+tree.cpp.

References printTree(), and root.

### 3.3.4 Member Data Documentation

#### 3.3.4.1 root

```
template<typename T>
LinkedBlock* BPlusTree< T >::root
```

Pointer to the root node of the B+ tree.

Definition at line 63 of file B+tree.cpp.

#### 3.3.4.2 t

```
template<typename T>
int BPlusTree< T >::t
```

Minimum degree of the B+ tree.

Defines the minimum and maximum number of keys in a node. Each node (except root) has at least `t-1` keys and at most `2*t-1` keys.

Definition at line 72 of file B+tree.cpp.

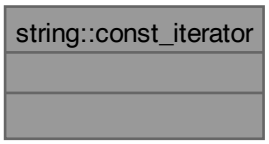
The documentation for this class was generated from the following file:

- source/B+tree.cpp

## 3.4 string::const_iterator Class Reference

STL iterator class.

Collaboration diagram for string::const_iterator:




### 3.4.1 Detailed Description

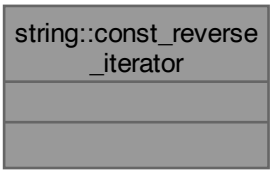
STL iterator class.

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

## 3.5 string::const_reverse_iterator Class Reference

STL iterator class.

Collaboration diagram for string::const_reverse_iterator:



### 3.5.1 Detailed Description

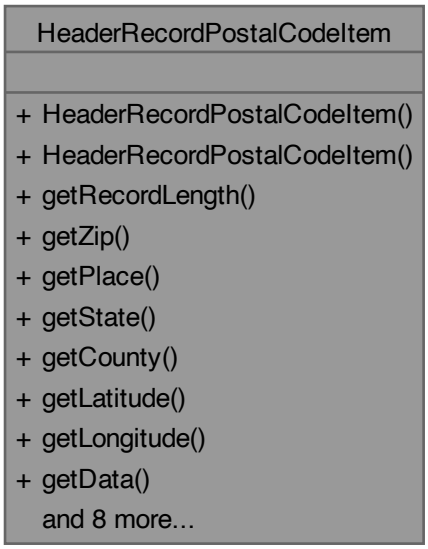
STL iterator class.

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

## 3.6 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.*

## 3.6.1 Detailed Description

Definition at line 16 of file HeaderRecordPostalCodeItem.h.

## 3.6.2 Constructor & Destructor Documentation

### 3.6.2.1 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.

#### 3.6.2.2 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.

### 3.6.3 Member Function Documentation

#### 3.6.3.1 getRecordLength()

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

Definition at line 64 of file HeaderRecordPostalCodeItem.cpp.

#### 3.6.3.2 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.

#### 3.6.3.3 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.

#### 3.6.3.4 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.

#### 3.6.3.5 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.

#### 3.6.3.6 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.

#### 3.6.3.7 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.

#### 3.6.3.8 getData()

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

Definition at line 127 of file HeaderRecordPostalCodeItem.cpp.

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

#### 3.6.3.9 setRecordLength()

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

Definition at line 133 of file HeaderRecordPostalCodeItem.cpp.

#### 3.6.3.10 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.

#### 3.6.3.11 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.

#### 3.6.3.12 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.

#### 3.6.3.13 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.

#### 3.6.3.14 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.

#### 3.6.3.15 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.

#### 3.6.3.16 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.

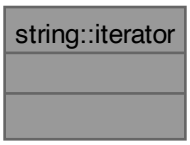
The documentation for this class was generated from the following files:

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

## 3.7 string::iterator Class Reference

STL iterator class.

Collaboration diagram for string::iterator:




### 3.7.1 Detailed Description

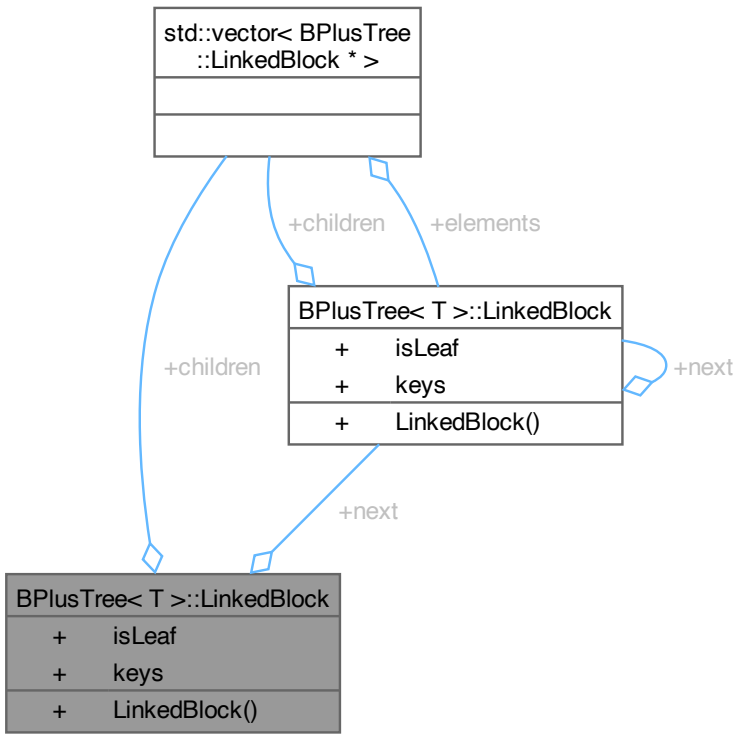
STL iterator class.

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

## 3.8 BPlusTree< T >::LinkedBlock Struct Reference

Node structure representing a B+ tree block.

Collaboration diagram for BPlusTree< T >::LinkedBlock:




### Public Member Functions

- LinkedBlock (bool leaf=false)

  *Constructs a LinkedBlock node.*

### Public Attributes

- bool isLeaf

  *Flag indicating whether this node is a leaf.*
- vector< T > keys

  *Keys stored in this node (sorted).*
- vector< LinkedBlock * > children

  *Child pointers.*
- LinkedBlock * next

  *Pointer to the next leaf node.*

### 3.8.1 Detailed Description

**template<typename T>**
**struct BPlusTree< T >::LinkedBlock**

Node structure representing a B+ tree block.

Each LinkedBlock can be either an internal node or a leaf. Leaf nodes are linked together using the `next` pointer to support efficient range queries.

Definition at line 27 of file B+tree.cpp.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 LinkedBlock()

```
template<typename T>
BPlusTree< T >::LinkedBlock::LinkedBlock (
            bool leaf = false)  [inline]
```

Constructs a LinkedBlock node.

**Parameters**

| | |
|---|---|
| *leaf* | True if the node should be a leaf, false otherwise. |

Definition at line 56 of file B+tree.cpp.

References isLeaf, and next.

### 3.8.3 Member Data Documentation

#### 3.8.3.1 isLeaf

```
template<typename T>
bool BPlusTree< T >::LinkedBlock::isLeaf
```

Flag indicating whether this node is a leaf.

Definition at line 30 of file B+tree.cpp.

#### 3.8.3.2 keys

```
template<typename T>
vector<T> BPlusTree< T >::LinkedBlock::keys
```

Keys stored in this node (sorted).

Definition at line 33 of file B+tree.cpp.

#### 3.8.3.3 children

```
template<typename T>
vector<LinkedBlock *> BPlusTree< T >::LinkedBlock::children
```

Child pointers.

For internal nodes, this holds pointers to child nodes. For leaf nodes, this vector is typically empty.

Definition at line 41 of file B+tree.cpp.

#### 3.8.3.4 next

```
template<typename T>
LinkedBlock* BPlusTree< T >::LinkedBlock::next
```

Pointer to the next leaf node.

Used only when this node is a leaf. Supports fast traversal for range queries.

Definition at line 49 of file B+tree.cpp.

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

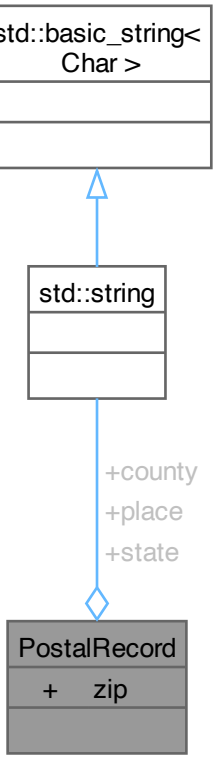
- source/B+tree.cpp

## 3.9 PostalRecord Struct Reference

Represents a record containing postal region information.

```
#include <PostalRecord.h>
```

Collaboration diagram for PostalRecord:




**Public Attributes**

- int zip

  *ZIP code (5-digit integer).*
- std::string place

  *Name of the city or locality associated with the ZIP code.*
- std::string state

  *U.S.*
- std::string county

  *County name associated with the ZIP code.*

### 3.9.1 Detailed Description

Represents a record containing postal region information.

The PostalRecord struct stores key postal attributes such as ZIP code, place name, state, and county. It can be used as a basic data container for mapping or searching postal information.

Definition at line 24 of file PostalRecord.h.

### 3.9.2 Member Data Documentation

#### 3.9.2.1 zip

```
int PostalRecord::zip
```

ZIP code (5-digit integer).

Definition at line 29 of file PostalRecord.h.

#### 3.9.2.2 place

```
std::string PostalRecord::place
```

Name of the city or locality associated with the ZIP code.

Definition at line 34 of file PostalRecord.h.

#### 3.9.2.3 state

```
std::string PostalRecord::state
```

U.S.

state abbreviation (e.g., "MN", "CA").

Definition at line 39 of file PostalRecord.h.

#### 3.9.2.4 county

```
std::string PostalRecord::county
```

County name associated with the ZIP code.

Definition at line 44 of file PostalRecord.h.

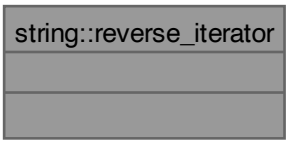
The documentation for this struct was generated from the following file:

- source/PostalRecord.h

## 3.10 string::reverse_iterator Class Reference

STL iterator class.

Collaboration diagram for string::reverse_iterator:



### 3.10.1 Detailed Description

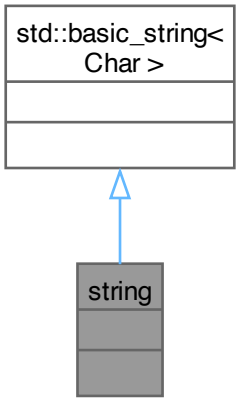
STL iterator class.

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

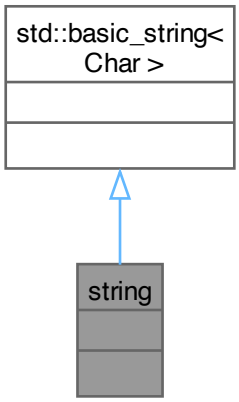
## 3.11 string Class Reference

STL class.

Inheritance diagram for string:




Collaboration diagram for string:



**Classes**

- class const_iterator

  *STL iterator class.*
- class const_reverse_iterator

  *STL iterator class.*
- class iterator

  *STL iterator class.*
- class reverse_iterator

  *STL iterator class.*

### 3.11.1 Detailed Description

STL class.

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

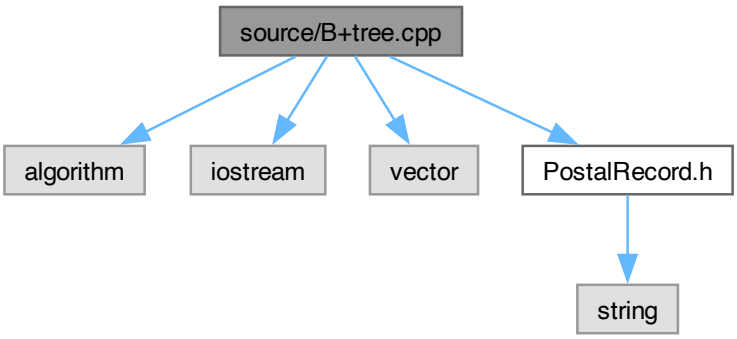
# Chapter 4

# File Documentation

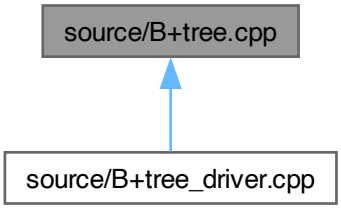
## 4.1 source/B+tree.cpp File Reference

```
#include <algorithm>
#include <iostream>
#include <vector>
#include "PostalRecord.h"
```

Include dependency graph for B+tree.cpp:



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




### Classes

- class BPlusTree< T >

  *B+ tree class template.*
- struct BPlusTree< T >::LinkedBlock

  *Node structure representing a B+ tree block.*

## 4.2 B+tree.cpp

Go to the documentation of this file.

```
00001 #include <algorithm>
00002 #include <iostream>
00003 #include <vector>
00004 #include "PostalRecord.h"
00005 using namespace std;
00006
00016 template <typename T>
00017 class BPlusTree
00018 {
00019 public:
00027     struct LinkedBlock
00028     {
00030         bool isLeaf;
00031
00033         vector<T> keys;
00034
00041         vector<LinkedBlock *> children;
00042
00049         LinkedBlock *next;
00050
00056         LinkedBlock(bool leaf = false)
00057             : isLeaf(leaf), next(nullptr)
00058         {
00059         }
00060     };
00061
00063     LinkedBlock *root;
00064
00072     int t;
00073
00084     void splitChild(LinkedBlock *parent, int index, LinkedBlock *child);
00085
00094     void insertNonFull(LinkedBlock *linkedBlock, T key);
00095
00104     void remove(LinkedBlock *linkedBlock, T key);
00105
00115     void borrowFromPrev(LinkedBlock *linkedBlock, int index);
00116
00126     void borrowFromNext(LinkedBlock *linkedBlock, int index);
00127
```

```
00137     void merge(LinkedBlock *linkedBlock, int index);
00138
00147     void printTree(LinkedBlock *linkedBlock, int level);
00148
00149 public:
00155     BPlusTree(int degree) : root(nullptr), t(degree) {}
00156
00164     void insert(T key);
00165
00173     bool search(T key);
00174
00182     void remove(T key);
00183
00193     vector<T> rangeQuery(T lower, T upper);
00194
00200     void printTree();
00201 };
00202
00203 // Implementation of splitChild function
00209 template <typename T>
00210 void BPlusTree<T>::splitChild(LinkedBlock *parent, int index,
00211                               LinkedBlock *child)
00212 {
00213     LinkedBlock *newChild = new LinkedBlock(child->isLeaf);
00214     parent->children.insert(
00215         parent->children.begin() + index + 1, newChild);
00216     parent->keys.insert(parent->keys.begin() + index,
00217                         child->keys[t - 1]);
00218
00219     newChild->keys.assign(child->keys.begin() + t,
00220                           child->keys.end());
00221     child->keys.resize(t - 1);
00222
00223     if (!child->isLeaf)
00224     {
00225         newChild->children.assign(child->children.begin() + t,
00226                                   child->children.end());
00227         child->children.resize(t);
00228     }
00229
00230     if (child->isLeaf)
00231     {
00232         newChild->next = child->next;
00233         child->next = newChild;
00234     }
00235 }
00236
00237 // Implementation of insertNonFull function
00243 template <typename T>
00244 void BPlusTree<T>::insertNonFull(LinkedBlock *linkedBlock, T key)
00245 {
00246     if (linkedBlock->isLeaf)
00247     {
00248         linkedBlock->keys.insert(upper_bound(linkedBlock->keys.begin(),
00249                                              linkedBlock->keys.end(),
00250                                              key),
00251                                  key);
00252     }
00253     else
00254     {
00255         int i = linkedBlock->keys.size() - 1;
00256         while (i >= 0 && key < linkedBlock->keys[i])
00257         {
00258             i--;
00259         }
00260         i++;
00261         if (linkedBlock->children[i]->keys.size() == 2 * t - 1)
00262         {
00263             splitChild(linkedBlock, i, linkedBlock->children[i]);
00264             if (key > linkedBlock->keys[i])
00265             {
00266                 i++;
00267             }
00268         }
00269         insertNonFull(linkedBlock->children[i], key);
00270     }
00271 }
00272
00273 // Implementation of remove function (internal helper)
00279 template <typename T>
00280 void BPlusTree<T>::remove(LinkedBlock *linkedBlock, T key)
00281 {
00282     // If linkedBlock is a leaf
00283     if (linkedBlock->isLeaf)
00284     {
00285         auto it = find(linkedBlock->keys.begin(), linkedBlock->keys.end(),
00286                        key);
```

```
00287         if (it != linkedBlock->keys.end())
00288         {
00289             linkedBlock->keys.erase(it);
00290         }
00291     }
00292     else
00293     {
00294         int idx = lower_bound(linkedBlock->keys.begin(),
00295                               linkedBlock->keys.end(), key) -
00296                   linkedBlock->keys.begin();
00297         if (idx < linkedBlock->keys.size() && linkedBlock->keys[idx] == key)
00298         {
00299             if (linkedBlock->children[idx]->keys.size() >= t)
00300             {
00301                 LinkedBlock *predLinkedBlock = linkedBlock->children[idx];
00302                 while (!predLinkedBlock->isLeaf)
00303                 {
00304                     predLinkedBlock = predLinkedBlock->children.back();
00305                 }
00306                 T pred = predLinkedBlock->keys.back();
00307                 linkedBlock->keys[idx] = pred;
00308                 remove(linkedBlock->children[idx], pred);
00309             }
00310             else if (linkedBlock->children[idx + 1]->keys.size() >= t)
00311             {
00312                 LinkedBlock *succLinkedBlock = linkedBlock->children[idx + 1];
00313                 while (!succLinkedBlock->isLeaf)
00314                 {
00315                     succLinkedBlock = succLinkedBlock->children.front();
00316                 }
00317                 T succ = succLinkedBlock->keys.front();
00318                 linkedBlock->keys[idx] = succ;
00319                 remove(linkedBlock->children[idx + 1], succ);
00320             }
00321             else
00322             {
00323                 merge(linkedBlock, idx);
00324                 remove(linkedBlock->children[idx], key);
00325             }
00326         }
00327         else
00328         {
00329             if (linkedBlock->children[idx]->keys.size() < t)
00330             {
00331                 if (idx > 0 && linkedBlock->children[idx - 1]->keys.size() >= t)
00332                 {
00333                     borrowFromPrev(linkedBlock, idx);
00334                 }
00335                 else if (idx < linkedBlock->children.size() - 1 && linkedBlock->children[idx + 1]
00336                                                                     ->keys.size() >= t)
00337                 {
00338                     borrowFromNext(linkedBlock, idx);
00339                 }
00340                 else
00341                 {
00342                     if (idx < linkedBlock->children.size() - 1)
00343                     {
00344                         merge(linkedBlock, idx);
00345                     }
00346                     else
00347                     {
00348                         merge(linkedBlock, idx - 1);
00349                     }
00350                 }
00351             }
00352             remove(linkedBlock->children[idx], key);
00353         }
00354     }
00355 }
00356
00357 // Implementation of borrowFromPrev function
00363 template <typename T>
00364 void BPlusTree<T>::borrowFromPrev(LinkedBlock *linkedBlock, int index)
00365 {
00366     LinkedBlock *child = linkedBlock->children[index];
00367     LinkedBlock *sibling = linkedBlock->children[index - 1];
00368
00369     child->keys.insert(child->keys.begin(),
00370                        linkedBlock->keys[index - 1]);
00371     linkedBlock->keys[index - 1] = sibling->keys.back();
00372     sibling->keys.pop_back();
00373
00374     if (!child->isLeaf)
00375     {
00376         child->children.insert(child->children.begin(),
00377                                sibling->children.back());
00378         sibling->children.pop_back();
```

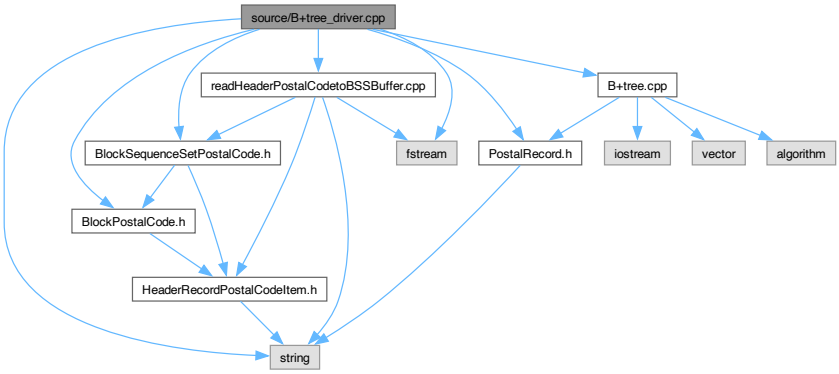
```
00379     }
00380 }
00381
00382 // Implementation of borrowFromNext function
00388 template <typename T>
00389 void BPlusTree<T>::borrowFromNext(LinkedBlock *linkedBlock, int index)
00390 {
00391     LinkedBlock *child = linkedBlock->children[index];
00392     LinkedBlock *sibling = linkedBlock->children[index + 1];
00393
00394     child->keys.push_back(linkedBlock->keys[index]);
00395     linkedBlock->keys[index] = sibling->keys.front();
00396     sibling->keys.erase(sibling->keys.begin());
00397
00398     if (!child->isLeaf)
00399     {
00400         child->children.push_back(
00401             sibling->children.front());
00402         sibling->children.erase(sibling->children.begin());
00403     }
00404 }
00405
00406 // Implementation of merge function
00412 template <typename T>
00413 void BPlusTree<T>::merge(LinkedBlock *linkedBlock, int index)
00414 {
00415     LinkedBlock *child = linkedBlock->children[index];
00416     LinkedBlock *sibling = linkedBlock->children[index + 1];
00417
00418     child->keys.push_back(linkedBlock->keys[index]);
00419     child->keys.insert(child->keys.end(),
00420                        sibling->keys.begin(),
00421                        sibling->keys.end());
00422     if (!child->isLeaf)
00423     {
00424         child->children.insert(child->children.end(),
00425                                sibling->children.begin(),
00426                                sibling->children.end());
00427     }
00428
00429     linkedBlock->keys.erase(linkedBlock->keys.begin() + index);
00430     linkedBlock->children.erase(linkedBlock->children.begin() + index + 1);
00431
00432     delete sibling;
00433 }
00434
00435 // Implementation of printTree function
00441 template <typename T>
00442 void BPlusTree<T>::printTree(LinkedBlock *linkedBlock, int level)
00443 {
00444     if (linkedBlock != nullptr)
00445     {
00446         for (int i = 0; i < level; ++i)
00447         {
00448             cout « "   ";
00449         }
00450         for (const T &key : linkedBlock->keys)
00451         {
00452             cout « key « " ";
00453         }
00454         cout « endl;
00455         for (LinkedBlock *child : linkedBlock->children)
00456         {
00457             printTree(child, level + 1);
00458         }
00459     }
00460 }
00461
00462 // Implementation of printTree wrapper function
00466 template <typename T>
00467 void BPlusTree<T>::printTree()
00468 {
00469     printTree(root, 0);
00470 }
00471
00472 // Implementation of search function
00478 template <typename T>
00479 bool BPlusTree<T>::search(T key)
00480 {
00481     LinkedBlock *current = root;
00482     while (current != nullptr)
00483     {
00484         int i = 0;
00485         while (i < current->keys.size() && key > current->keys[i])
00486         {
00487             i++;
00488         }
```

```
00489         if (i < current->keys.size() && key == current->keys[i])
00490         {
00491             return true;
00492         }
00493         if (current->isLeaf)
00494         {
00495             return false;
00496         }
00497         current = current->children[i];
00498     }
00499     return false;
00500 }
00501
00502 // Implementation of range query function
00511 template <typename T>
00512 vector<T> BPlusTree<T>::rangeQuery(T lower, T upper)
00513 {
00514     vector<T> result;
00515     LinkedBlock *current = root;
00516     while (!current->isLeaf)
00517     {
00518         int i = 0;
00519         while (i < current->keys.size() && lower > current->keys[i])
00520         {
00521             i++;
00522         }
00523         current = current->children[i];
00524     }
00525     while (current != nullptr)
00526     {
00527         for (const T &key : current->keys)
00528         {
00529             if (key >= lower && key <= upper)
00530             {
00531                 result.push_back(key);
00532             }
00533             if (key > upper)
00534             {
00535                 return result;
00536             }
00537         }
00538         current = current->next;
00539     }
00540     return result;
00541 }
00542
00543 // Implementation of insert function
00549 template <typename T>
00550 void BPlusTree<T>::insert(T key)
00551 {
00552     if (root == nullptr)
00553     {
00554         root = new LinkedBlock(true);
00555         root->keys.push_back(key);
00556     }
00557     else
00558     {
00559         if (root->keys.size() == 2 * t - 1)
00560         {
00561             LinkedBlock *newRoot = new LinkedBlock();
00562             newRoot->children.push_back(root);
00563             splitChild(newRoot, 0, root);
00564             root = newRoot;
00565         }
00566         insertNonFull(root, key);
00567     }
00568 }
00569
00570 // Implementation of remove function (public)
00576 template <typename T>
00577 void BPlusTree<T>::remove(T key)
00578 {
00579     if (root == nullptr)
00580     {
00581         return;
00582     }
00583     remove(root, key);
00584     if (root->keys.empty() && !root->isLeaf)
00585     {
00586         LinkedBlock *tmp = root;
00587         root = root->children[0];
00588         delete tmp;
00589     }
00590 }
```

# 4.3 source/B+tree_driver.cpp File Reference

```
#include <string>
#include "BlockPostalCode.h"
#include "BlockSequenceSetPostalCode.h"
#include "PostalRecord.h"
#include "readHeaderPostalCodetoBSSBuffer.cpp"
#include <fstream>
#include "B+tree.cpp"
```

Include dependency graph for B+tree_driver.cpp:




### Functions

- bool lookupPostalRecord (int zip, PostalRecord &out, BlockSequenceSetPostalCode &bss)

  *Searches the Block Sequence Set (BSS) for a postal record by ZIP.*
- int main ()

  *Main function: builds B+ tree from postal codes and performs lookup.*

## 4.3.1 Function Documentation

### 4.3.1.1 lookupPostalRecord()

```
bool lookupPostalRecord (
            int zip,
            PostalRecord & out,
            BlockSequenceSetPostalCode & bss)
```

Searches the Block Sequence Set (BSS) for a postal record by ZIP.

Performs sequential traversal of the BSS linked blocks.
Only used *after* the B+ tree confirms the ZIP exists.

**Parameters**

| | |
|---|---|
| *zip* | ZIP code to search for. |
| *out* | Reference to PostalRecord where the result will be stored. |
| *bss* | Reference to the Block Sequence Set containing postal blocks. |

**Returns**

true If the ZIP was found in the sequence set.

false If the ZIP does not exist in the sequence set.

Definition at line 35 of file B+tree_driver.cpp.

References PostalRecord::county, BlockPostalCode::getBlockItem(), HeaderRecordPostalCodeItem::getCounty(), BlockSequenceSetPostalCode::getHead(), BlockPostalCode::getNext(), HeaderRecordPostalCodeItem::getPlace(), HeaderRecordPostalCodeItem::getState(), HeaderRecordPostalCodeItem::getZip(), PostalRecord::place, PostalRecord::state, and PostalRecord::zip.

#### 4.3.1.2 main()

```
int main ()
```

Main function: builds B+ tree from postal codes and performs lookup.

Steps:

1. Builds a Block Sequence Set from a length-indicated record file.
2. Inserts ZIP codes from each block into a B+ tree.
3. Prints the B+ tree structure to `B+Tree_data.txt`.
4. Performs user-driven ZIP lookups using:
   - B+ tree (index check)
   - Sequence set (record retrieval)

**Returns**

int Program exit code.

< B+ tree degree. Higher degree → shorter tree height.

< Input postal data file

< Output dump containing tree structure

< Sequence set structure for all blocks

Loads all postal header+records into the Block Sequence Set.

< First block pointer

Insert ZIP codes from all intermediate blocks.

Prints the B+ tree in a readable hierarchical structure to file.

User search loop for interactive ZIP lookup.

Definition at line 81 of file B+tree_driver.cpp.

References PostalRecord::county, BlockPostalCode::getBlockItem(), BlockSequenceSetPostalCode::getHead(), BlockPostalCode::getNext(), HeaderRecordPostalCodeItem::getZip(), inputDatatoBlockSequenceSet(), BPlusTree< T >::insert(), lookupPostalRecord(), PostalRecord::place, BPlusTree< T >::printTree(), BPlusTree< T >::search(), PostalRecord::state, and PostalRecord::zip.

# 4.4 B+tree_driver.cpp

Go to the documentation of this file.

```
00001
00011
00012 #include <string>
00013 #include "BlockPostalCode.h"
00014 #include "BlockSequenceSetPostalCode.h"
00015 #include "PostalRecord.h"
00016 #include "readHeaderPostalCodetoBSSBuffer.cpp"
00017 #include <fstream>
00018
00019 #include "B+tree.cpp"
00020
00021 using namespace std;
00022
00035 bool lookupPostalRecord(int zip,
00036                         PostalRecord &out,
00037                         BlockSequenceSetPostalCode &bss)
00038 {
00039     // Start with the head block (by value, just like in main)
00040     BlockPostalCode current = bss.getHead();
00041
00042     while (true)
00043     {
00044         HeaderRecordPostalCodeItem item = current.getBlockItem();
00045
00046         if (item.getZip() == zip)
00047         {
00048             out.zip = item.getZip();
00049             out.place = item.getPlace();
00050             out.state = item.getState();
00051             out.county = item.getCounty();
00052             return true;
00053         }
00054
00055         // If there's no "next" block, we're done
00056         if (current.getNext() == nullptr)
00057         {
00058             break;
00059         }
00060
00061         // Move to the next block (getNext() returns BlockPostalCode*)
00062         current = *current.getNext();
00063     }
00064
00065     return false; // not found in the sequence set
00066 }
00067
00081 int main()
00082 {
00083     int degree = 10;
00084     BPlusTree<int> tree(degree);
00085
00086     string fileName = "us_postal_codes_length_indicated_header_record.txt";
00087     ofstream outputFile("B+Tree_data.txt");
00088
00089     BlockSequenceSetPostalCode myBlockSequenceSetPostalCode;
00090     string blockRecord;
00091     BlockPostalCode myBlock;
00092
00096     inputDatatoBlockSequenceSet(myBlockSequenceSetPostalCode, fileName);
00097
00098     myBlock = myBlockSequenceSetPostalCode.getHead();
00099
00100     // Insert first block ZIP into B+ tree
00101     tree.insert(myBlock.getBlockItem().getZip());
00102
00103     // Move to next block
00104     myBlock = *myBlock.getNext();
00105
00109     while (myBlock.getNext() != nullptr)
00110     {
00111         tree.insert(myBlock.getBlockItem().getZip());
00112         myBlock = *myBlock.getNext();
00113     }
00114
00115     // Insert last block ZIP
00116     tree.insert(myBlock.getBlockItem().getZip());
00117
00118     // Save original std::cout buffer
00119     std::streambuf *originalCoutBuffer = std::cout.rdbuf();
00120
00121     // Redirect std::cout to the output file
00122     std::cout.rdbuf(outputFile.rdbuf());
```

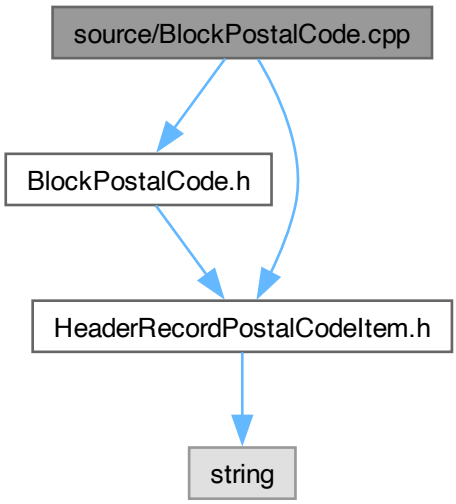
```
00123
00127     tree.printTree();
00128
00129     // Restore cout output
00130     std::cout.rdbuf(originalCoutBuffer);
00131
00132     outputFile.close();
00133
00134     cout « "B+tree builded successfully!" « endl;
00135     cout « "B+ tree file: B+Tree_data.txt" « endl;
00136
00140     int zip;
00141     while (true)
00142     {
00143         std::cout « "Enter ZIP to search (0 to quit): ";
00144         if (!(std::cin » zip))
00145         {
00146             std::cout « "Input error, exiting...\n";
00147             break;
00148         }
00149
00150         if (zip == 0)
00151         {
00152             break;
00153         }
00154
00155         // B+ tree check
00156         if (tree.search(zip))
00157         {
00158             PostalRecord rec;
00159
00160             // If ZIP exists, retrieve full record from sequence set
00161             if (lookupPostalRecord(zip, rec, myBlockSequenceSetPostalCode))
00162             {
00163                 std::cout « "\nFOUND ZIP " « rec.zip « "\n"
00164                           « "Place:  " « rec.place « "\n"
00165                           « "State:  " « rec.state « "\n"
00166                           « "County: " « rec.county « "\n\n";
00167             }
00168             else
00169             {
00170                 std::cout « "ZIP " « zip
00171                           « " FOUND in B+ tree\n\n";
00172             }
00173         }
00174         else
00175         {
00176             std::cout « "ZIP " « zip « " NOT FOUND in B+ tree\n\n";
00177         }
00178     }
00179
00180     return 0;
00181 }
```

## 4.5 source/BlockPostalCode.cpp File Reference

Implements the BlockPostalCode class.

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

Include dependency graph for BlockPostalCode.cpp:




### 4.5.1 Detailed Description

Implements the BlockPostalCode class.

Definition in file BlockPostalCode.cpp.

## 4.6 BlockPostalCode.cpp

Go to the documentation of this file.

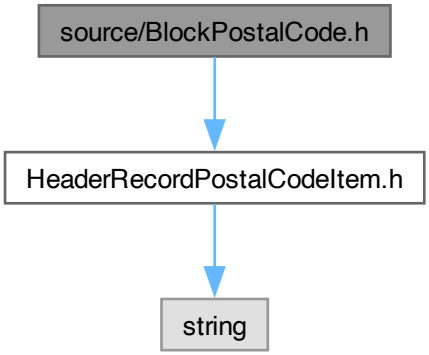
```
00001
00005
00006 #include "BlockPostalCode.h"
00007 #include "HeaderRecordPostalCodeItem.h"
00008
00012 BlockPostalCode::BlockPostalCode() : prevRBN(nullptr), nextRBN(nullptr) {}
00013
00018 BlockPostalCode::BlockPostalCode(const HeaderRecordPostalCodeItem &item) : data(item),
      prevRBN(nullptr), 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 }
```

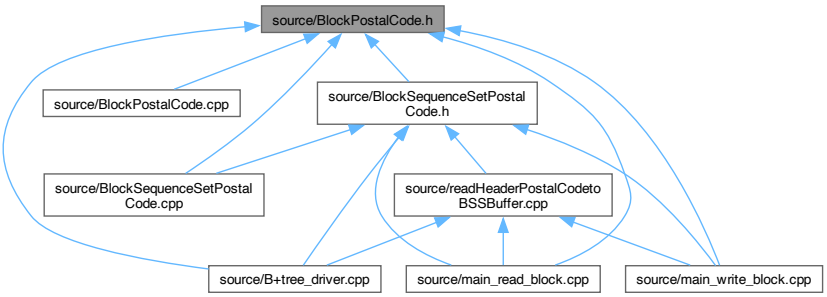
## 4.7 source/BlockPostalCode.h File Reference

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

`#include "HeaderRecordPostalCodeItem.h"`
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.*

### 4.7.1 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.

## 4.8 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 #endif
```

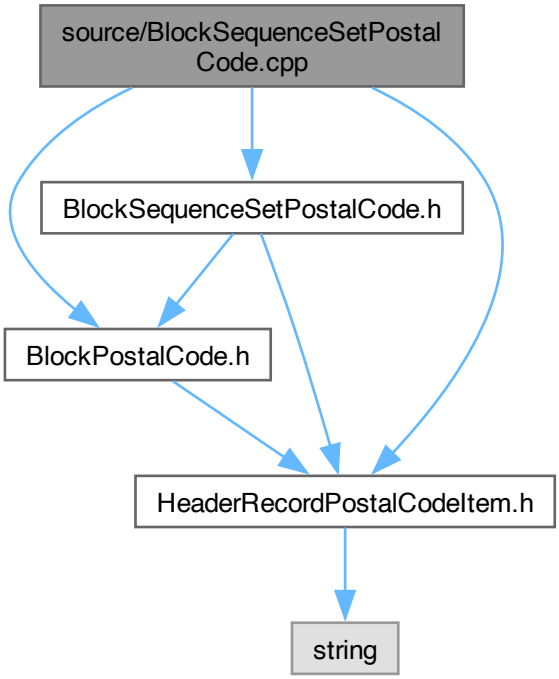
## 4.9 source/BlockSequenceSetPostalCode.cpp File Reference

Implements the BlockSequenceSetPostalCode class.

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

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

Include dependency graph for BlockSequenceSetPostalCode.cpp:




### 4.9.1 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.

## 4.10 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
```

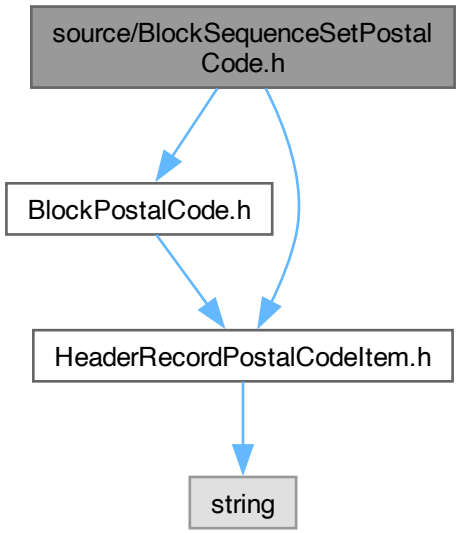
```
00036 {
00037     return *headBlock;
00038 }
00039
00051 bool BlockSequenceSetPostalCode::add(const HeaderRecordPostalCodeItem &newHeaderPostalCodeItem)
00052 {
00053     BlockPostalCode *newBlock = new BlockPostalCode();
00054     newBlock->setBlockItem(newHeaderPostalCodeItem);
00055
00056     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 }
```

## 4.11 source/BlockSequenceSetPostalCode.h File Reference

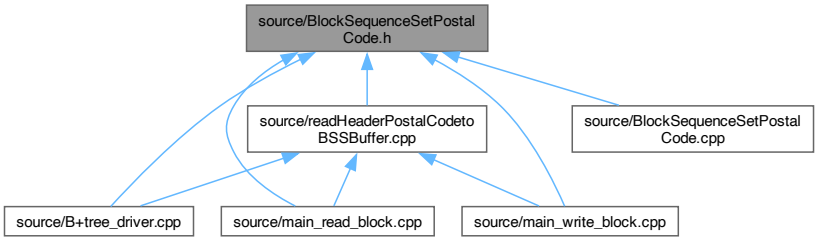
Declares the BlockSequenceSetPostalCode class which manages a doubly linked sequence of BlockPostalCode nodes.

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

Include 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.*

### 4.11.1 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.

## 4.12 BlockSequenceSetPostalCode.h

Go to the documentation of this file.

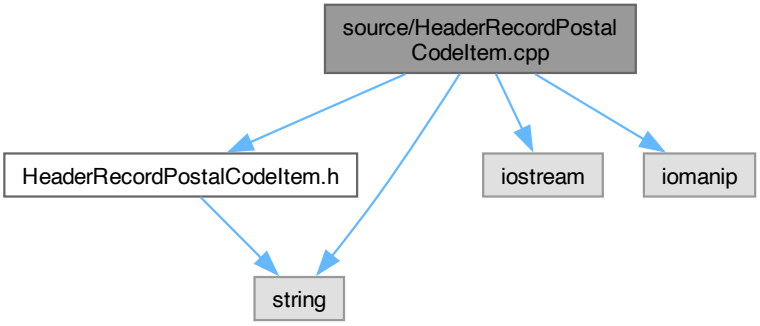
```
00001 #ifndef BLOCK_SEQUENCE_SET_POSTAL_CODE
00002 #define BLOCK_SEQUENCE_SET_POSTAL_CODE
00003
00013
00014 #include "BlockPostalCode.h"
00015 #include "HeaderRecordPostalCodeItem.h"
00016
00035 class BlockSequenceSetPostalCode
00036 {
00037 private:
00038     BlockPostalCode *headBlock;
00039     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 #endif
```

## 4.13 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:




### 4.13.1 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.

## 4.14 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()
```

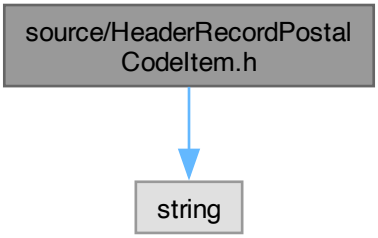
```
00032 {
00033     recordLength = 0;
00034     zip = 0;
00035     place = "";
00036     state = "";
00037     county = "";
00038     latitude = 0;
00039     longitude = 0;
00040 }
00041
00053 HeaderRecordPostalCodeItem::HeaderRecordPostalCodeItem(int r, int z, const string &p, const string &s,
      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 }
```

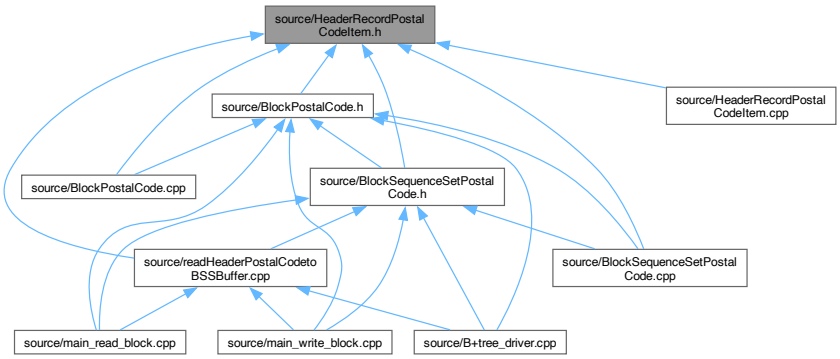
## 4.15 source/HeaderRecordPostalCodeItem.h File Reference

Defines and manages individual HeaderRecord postal code items.

`#include <string>`
Include dependency graph for HeaderRecordPostalCodeItem.h:




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



**Classes**

- class HeaderRecordPostalCodeItem
- class string

  *STL class.*

### 4.15.1 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.

## 4.16 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 #include <string>
00012 using std::string;
00013
00014 using namespace std;
00015
00016 class HeaderRecordPostalCodeItem
00017 {
00018 private:
00019     int recordLength;
00020     int zip;
00021     string place;
00022     string state;
00023     string county;
00024     double latitude;
00025     double longitude;
00026
00027 public:
00044     HeaderRecordPostalCodeItem();
00045
00057     HeaderRecordPostalCodeItem(int r, int z, const string &p, const string &s, const string &c, double
      lat, double lon);
00058
00059     int getRecordLength() const;
00060
00065     int getZip() const;
00066
00071     string getPlace() const;
00072
00077     string getState() const;
00078
00085     string getCounty() const;
00086
00092     double getLatitude() const;
00093
00099     double getLongitude() const;
00100
00101     string getData() const;
00102
```

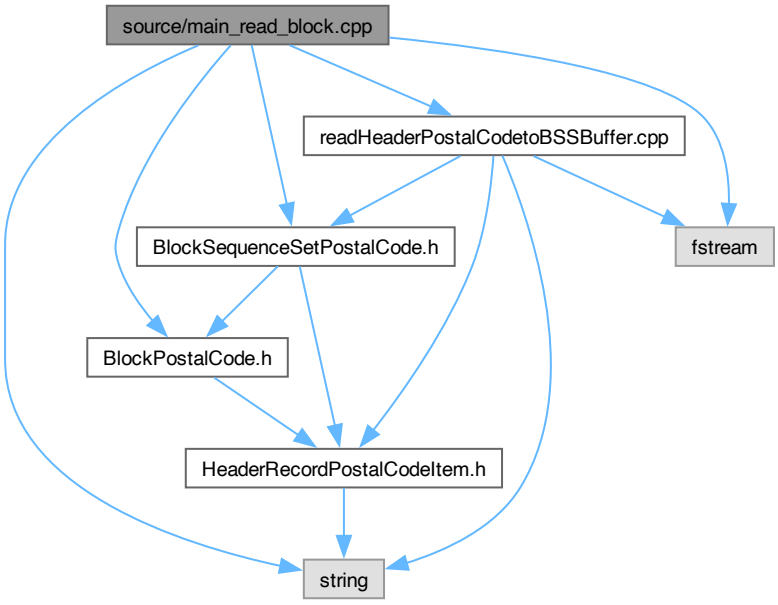
```
00103     void setRecordLength(int newRecordLength);
00104
00110     void setZip(int newZip);
00111
00117     void setPlace(const string &newPlace);
00118
00124     void setState(const string &newState);
00125
00131     void setCounty(const string &newCounty);
00132
00139     void setLatitude(double newLat);
00140
00147     void setLongitude(double newLon);
00148
00155     void printInfo() const;
00156 };
00157
00158 #endif
```

## 4.17 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.*

### 4.17.1 Detailed Description

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

Definition in file main_read_block.cpp.

### 4.17.2 Function Documentation

#### 4.17.2.1 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().

## 4.18 main_read_block.cpp

Go to the documentation of this file.

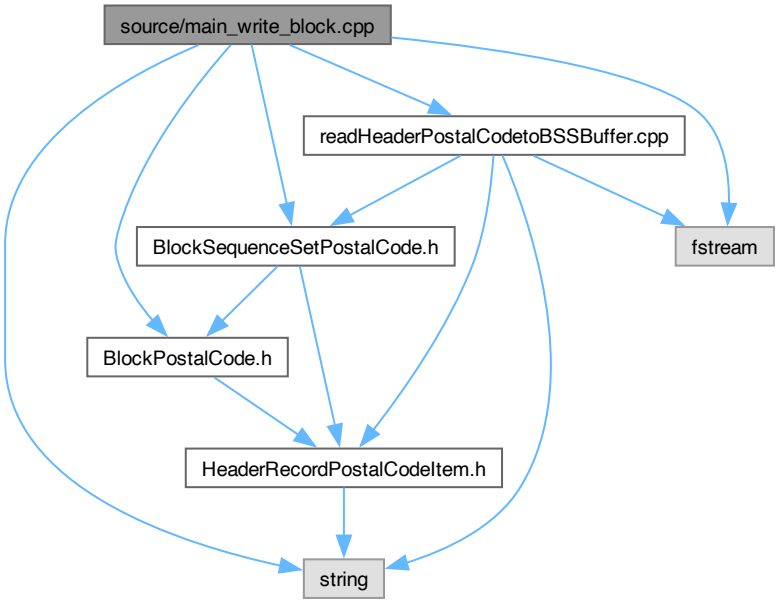
```
00001
00005
00006 #include <string>
00007 #include "BlockPostalCode.h"
00008 #include "BlockSequenceSetPostalCode.h"
00009 #include "readHeaderPostalCodetoBSSBuffer.cpp"
00010 #include <fstream>
00011
00012 using namespace std;
00013
00026 int main()
00027 {
00028     string fileName = "us_postal_codes_length_indicated_header_record.txt";
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 }
```

## 4.19 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.*

## 4.19.1 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.

## 4.19.2 Function Documentation

### 4.19.2.1 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().

## 4.20 main_write_block.cpp

Go to the documentation of this file.

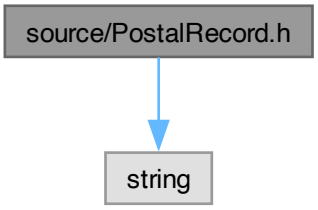
```
00001
00010
00011 #include <string>
00012 #include "BlockPostalCode.h"
00013 #include "BlockSequenceSetPostalCode.h"
00014 #include "readHeaderPostalCodetoBSSBuffer.cpp"
00015 #include <fstream>
00016
00017 using namespace std;
00018
00029 int main()
00030 {
00031     string fileName = "us_postal_codes_length_indicated_header_record.txt";
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 }
```

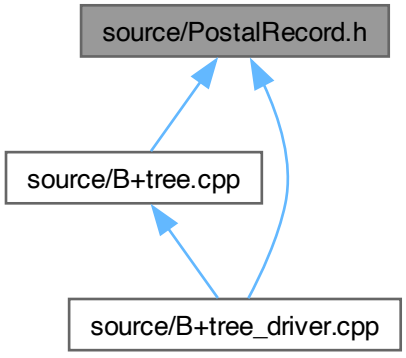
## 4.21 source/PostalRecord.h File Reference

Defines the PostalRecord struct used to store ZIP code information.

`#include <string>`
Include dependency graph for PostalRecord.h:




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



**Classes**

- struct PostalRecord

  *Represents a record containing postal region information.*

### 4.21.1 Detailed Description

Defines the PostalRecord struct used to store ZIP code information.

This file contains the definition of the PostalRecord structure, which holds information about a postal region, including the ZIP code, place name, state, and county. It is intended for use in applications that process or store geographic or postal data.

Definition in file PostalRecord.h.

## 4.22 PostalRecord.h

Go to the documentation of this file.

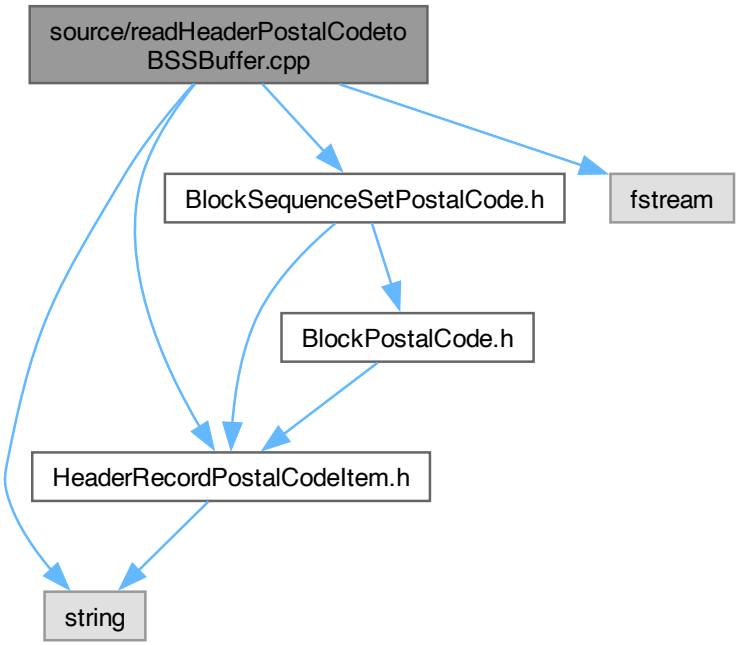
```
00001
00010
00011 #ifndef POSTAL_RECORD_H
00012 #define POSTAL_RECORD_H
00013
00014 #include <string>
00015
00024 struct PostalRecord
00025 {
00029     int zip;
00030
00034     std::string place;
00035
00039     std::string state;
00040
00044     std::string county;
00045 };
00046
00047 #endif // POSTAL_RECORD_H
```

## 4.23 source/readHeaderPostalCodetoBSSBuffer.cpp File Reference

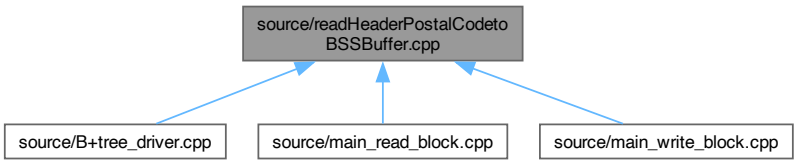
```
#include <string>
#include "HeaderRecordPostalCodeItem.h"
#include "BlockSequenceSetPostalCode.h"
```

```
#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)

## 4.23.1 Function Documentation

### 4.23.1.1 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().

## 4.24 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 }
```