Assignment 2

Generated by Doxygen 1.9.1

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:

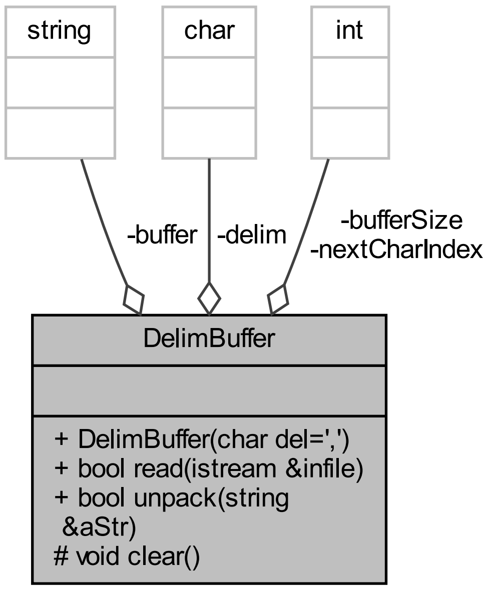
Chapter 3

Class Documentation

3.1 DelimBuffer Class Reference

```
#include <DelimBuffer.h>
```

Collaboration diagram for DelimBuffer:


string
char
int
-buffer
-delim
-bufferSize
-nextCharIndex
DelimBuffer
+ DelimBuffer(char del=',')
+ bool read(istream &infile)
+ bool unpack(string &aStr)
# void clear()

Public Member Functions

- DelimBuffer (char del=',')

  Default Constructor.
- bool read (istream &infile)

  method reads from the file stream and tests to see if stream open
- bool unpack (string &aStr)

  method "unpacks" the passed string pointer

Protected Member Functions

- void clear ()

  clear the buffer

Private Attributes

- char delim = ' '
- string buffer

  Delimiter.
- int bufferSize = 0

  The buffer.
- int nextCharIndex = 0

  Size of the buffer string.

3.1.1 Detailed Description

Definition at line 21 of file DelimBuffer.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DelimBuffer()

```
DelimBuffer::DelimBuffer (
             char del = ',' )  [inline]
```

Definition at line 26 of file DelimBuffer.h.

3.1.3 Member Function Documentation

3.1.3.1 clear()

```
void DelimBuffer::clear ( )  [protected]
```

Parameters

no	parameters passed

Returns

no return values

Definition at line 16 of file DelimBuffer.cpp.

3.1.3.2 read()

```
bool DelimBuffer::read (
             istream & infile )
```

Parameters

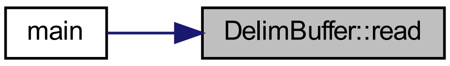
infile	file stream is passed in

Returns

returns a boolean value about whether the stream is open or not

Definition at line 25 of file DelimBuffer.cpp.

Here is the caller graph for this function:



3.1.3.3 unpack()

```
bool DelimBuffer::unpack (
             string & aStr )
```

Parameters

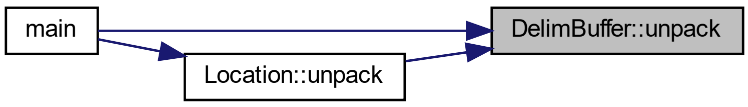
string	pointer aStr

Returns

Returns a boolean value if the delimeter is in the string or not

Definition at line 36 of file DelimBuffer.cpp.

Here is the caller graph for this function:



3.1.4 Member Data Documentation

3.1.4.1 buffer

```
string DelimBuffer::buffer  [private]
```

Definition at line 41 of file DelimBuffer.h.

3.1.4.2 bufferSize

```
int DelimBuffer::bufferSize = 0  [private]
```

Definition at line 42 of file DelimBuffer.h.

3.1.4.3 delim

```
char DelimBuffer::delim = ' '  [private]
```

Definition at line 40 of file DelimBuffer.h.

3.1.4.4 nextCharIndex

```
int DelimBuffer::nextCharIndex = 0  [private]
```

Definition at line 43 of file DelimBuffer.h.

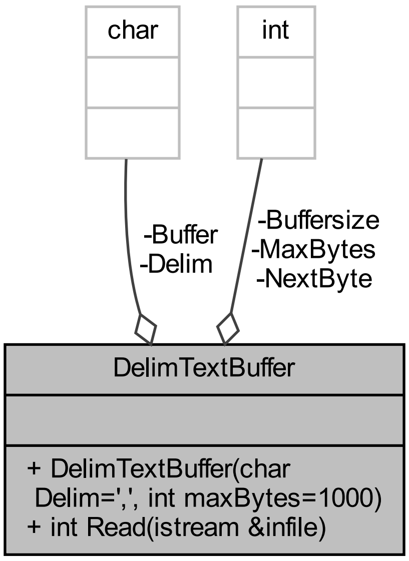
The documentation for this class was generated from the following files:

- DelimBuffer.h
- DelimBuffer.cpp

3.2 DelimTextBuffer Class Reference

```
#include <Buffer.h>
```

Collaboration diagram for DelimTextBuffer:


char
int
-Buffer
-Delim
-Buffersize
-MaxBytes
-NextByte
DelimTextBuffer
+ DelimTextBuffer(char Delim=',', int maxBytes=1000)
+ int Read(istream &infile)

Public Member Functions

- DelimTextBuffer (char Delim=',', int maxBytes=1000)
- int Read (istream &infile)

Private Attributes

- char Delim
- char ∗ Buffer
- int Buffersize
- int MaxBytes
- int NextByte

3.2.1 Detailed Description

Definition at line 8 of file Buffer.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DelimTextBuffer()

```
DelimTextBuffer::DelimTextBuffer (
             char Delim = ',',
             int maxBytes = 1000 )
```

3.2.3 Member Function Documentation

3.2.3.1 Read()

```
int DelimTextBuffer::Read (
             istream & infile )
```

Definition at line 9 of file Buffer.cpp.

3.2.4 Member Data Documentation

3.2.4.1 Buffer

```
char* DelimTextBuffer::Buffer  [private]
```

Definition at line 20 of file Buffer.h.

3.2.4.2 Buffersize

```
int DelimTextBuffer::Buffersize  [private]
```

Definition at line 21 of file Buffer.h.

3.2.4.3 Delim

```
char DelimTextBuffer::Delim  [private]
```

Definition at line 19 of file Buffer.h.

3.2.4.4 MaxBytes

```
int DelimTextBuffer::MaxBytes  [private]
```

Definition at line 22 of file Buffer.h.

3.2.4.5 NextByte

```
int DelimTextBuffer::NextByte  [private]
```

Definition at line 23 of file Buffer.h.

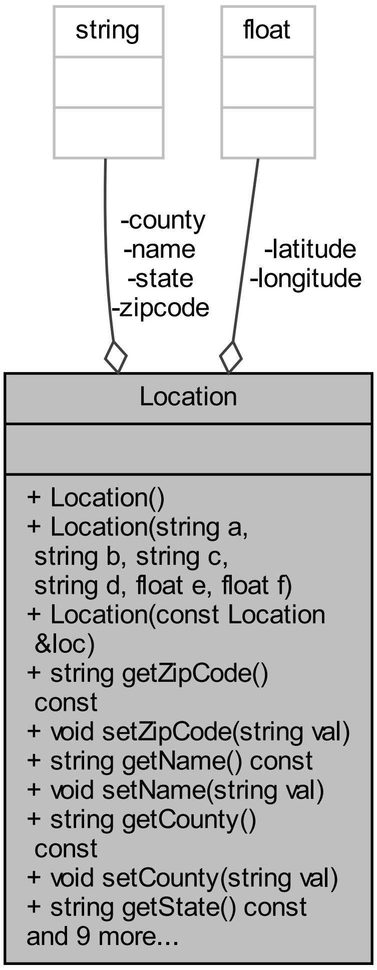
The documentation for this class was generated from the following files:

- Buffer.h
- Buffer.cpp

3.3 Location Class Reference

```
#include <Location.h>
```

Collaboration diagram for Location:


string
float
-county
-name
-state
-zipcode
-latitude
-longitude
Location
+ Location()
+ Location(string a, string b, string c, string d, float e, float f)
+ Location(const Location &loc)
+ string getZipCode() const
+ void setZipCode(string val)
+ string getName() const
+ void setName(string val)
+ string getCounty() const
+ void setCounty(string val)
+ string getState() const
and 9 more...

Public Member Functions

- Location ()
- Location (string a, string b, string c, string d, float e, float f)

  overloaded constructor
- Location (const Location &loc)

  copy constructor
- string getZipCode () const
- void setZipCode (string val)
- string getName () const
- void setName (string val)
- string getCounty () const
- void setCounty (string val)
- string getState () const
- void setState (string val)

- float getLat () const
- void setLat (float val)
- float getLong () const
- void setLong (float val)
- bool unpack (DelimBuffer &buffer)

  This method "unpacks" the delimiter object that is passed and reads from its provided file.
- void operator= (const Location &loc)

  overloaded assignment operator for a location object
- bool operator< (const Location &loc) const

  less than comparison operator overloaded
- bool operator> (const Location &loc) const

  less than comparison operator overloaded

Private Attributes

- string zipcode

  Member variable "zipcode".
- string name

  Member variable "name".
- string county

  Member variable "county".
- string state

  Member variable "state".
- float latitude

  Member variable "lat".
- float longitude

  Member variable "long".

Friends

- ostream & operator<< (ostream &out, const Location &loc)

  outstream operator overloaded

3.3.1 Detailed Description

Definition at line 21 of file Location.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Location() [1/3]

```
Location::Location ( )  [inline]
```

Default constructor

Definition at line 25 of file Location.h.

3.3.2.2 Location() [2/3]

```
Location::Location (
             string a,
             string b,
             string c,
             string d,
             float e,
             float f )   [inline]
```

Parameters

string	a sets zipcode
string	b sets name of place
string	c sets name of county
string	d sets two letter state code
float	e sets the latitude value
float	f sets the longitude value

Definition at line 42 of file Location.h.

3.3.2.3 Location() [3/3]

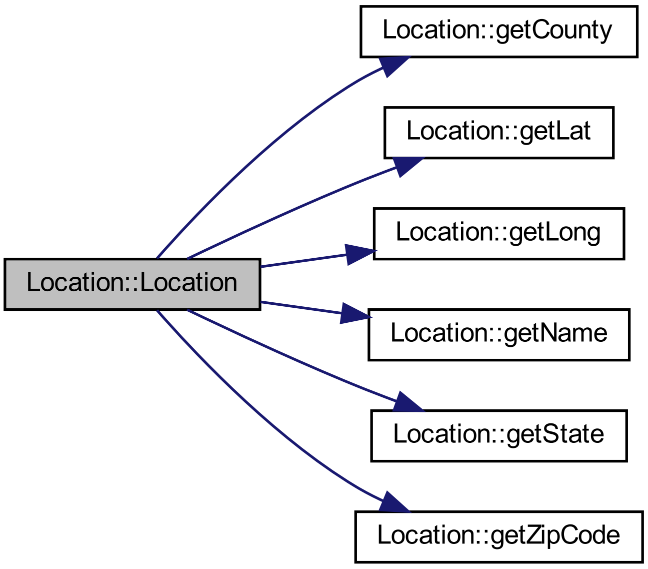
```
Location::Location (
             const Location & loc )   [inline]
```

Parameters

passes	a Location object to create another object

Definition at line 54 of file Location.h.

Here is the call graph for this function:



3.3.3 Member Function Documentation

3.3.3.1 getCounty()

`string Location::getCounty ( ) const`

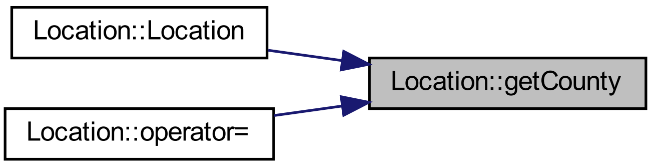
Access county

Returns

The current value of county

Definition at line 35 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.2 getLat()

```
float Location::getLat ( ) const
```

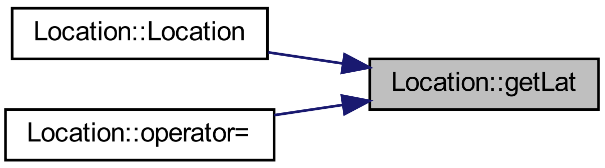
Access lat

Returns

The current Latitude value

Definition at line 37 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.3 getLong()

```
float Location::getLong ( ) const
```

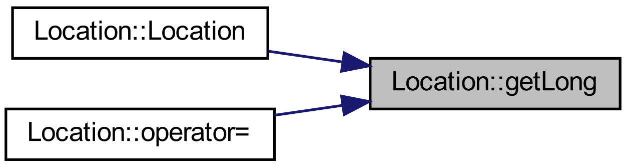
Access long

Returns

The current Longitude value

Definition at line 38 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.4 getName()

```
string Location::getName ( ) const
```

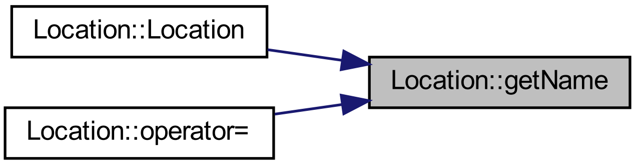
Access name

Returns

The current value of name

Definition at line 34 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.5 getState()

```
string Location::getState ( ) const
```

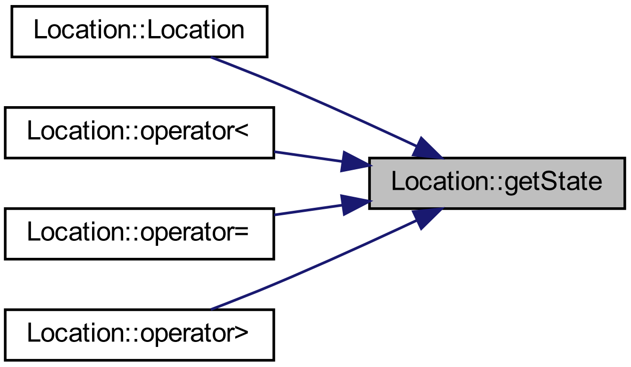
Access state

Returns

The current value of state

Definition at line 36 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.6 getZipCode()

```
string Location::getZipCode ( ) const
```

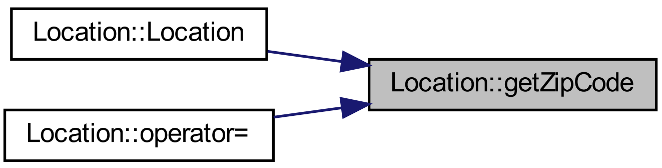
Access zipcode

Returns

The current value of zipcode

Definition at line 33 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.7 operator<()

```
bool Location::operator< (
            const Location & loc ) const
```

Parameters

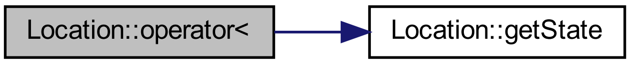
Location	class object

Returns

returns the overloaded operator's greater operand

Definition at line 95 of file Location.cpp.

Here is the call graph for this function:



3.3.3.8 operator=()

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

Parameters

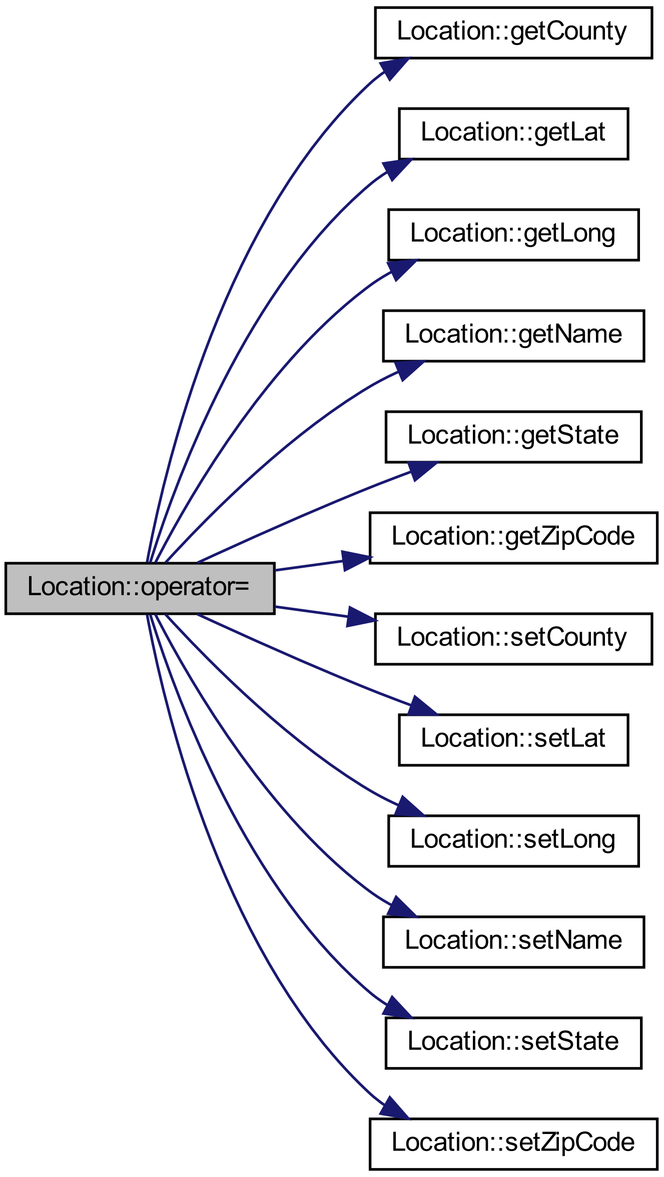
passed	a Location class object

Returns

no return value

Definition at line 67 of file Location.cpp.

Here is the call graph for this function:



3.3.3.9 operator>()

```
bool Location::operator> (
             const Location & loc ) const
```

Parameters

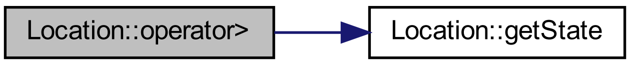
Location	class object

Returns

returns the overloaded operator's least operand

Definition at line 103 of file Location.cpp.

Here is the call graph for this function:



3.3.3.10 setCounty()

```
void Location::setCounty (
             string val )
```

Set county

Parameters

val	New value to set

Postcondition

sets val to County

Parameters

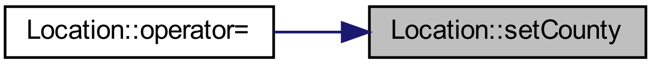
passed	a string value to set county name

Returns

no return value

Definition at line 50 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.11 setLat()

```
void Location::setLat (
            float val )
```

Set latitude

Parameters

val	New value to set

Postcondition

sets val to Latitude

Parameters

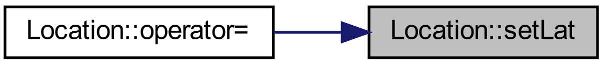
passed	a string value to set latitude

Returns

no return value

Definition at line 58 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.12 setLong()

```
void Location::setLong (
             float val )
```

Set long

Parameters

val	New value to set

Postcondition

sets val to Longitude

Parameters

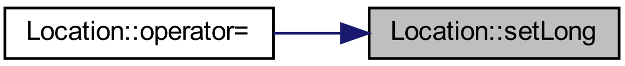
passed	a string value to set place longitude

Returns

no return value

Definition at line 62 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.13 setName()

```
void Location::setName (
             string val )
```

Set name

Parameters

val	New value to set

Postcondition

sets val to Name

Parameters

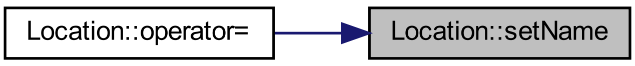
passed	a string value to set place name

Returns

no return value

Definition at line 46 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.14 setState()

```
void Location::setState (
             string val )
```

Set state

Parameters

val	New value to set

Postcondition

sets val to State

Parameters

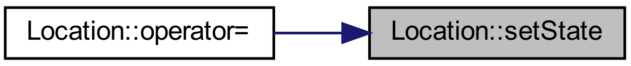
passed	a string value to set state name

Returns

no return value

Definition at line 54 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.15 setZipCode()

```
void Location::setZipCode (
             string val )
```

Set zipcode

Parameters

val	New value to set

Postcondition

sets val to the zipCode

Parameters

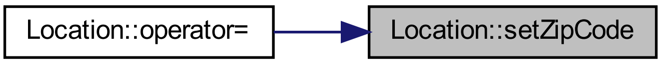
passed	a string value to set zipcode

Returns

no return value

Definition at line 42 of file Location.cpp.

Here is the caller graph for this function:



3.3.3.16 unpack()

```
bool Location::unpack (
            DelimBuffer & buffer )
```

Parameters

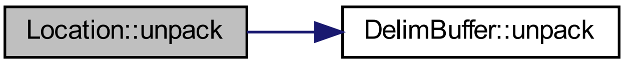
DelimBuffer	object buffer

Returns

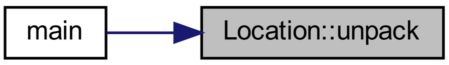
Returns a boolean result about whether the object piece is actually there

Definition at line 18 of file Location.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.4 Friends And Related Function Documentation

3.3.4.1 operator<<

```
ostream& operator<< (
             ostream & out,
             const Location & loc )  [friend]
```

Parameters

outstream	
Location	class object

Returns

returns the overloaded operator right hand operand

Definition at line 81 of file Location.cpp.

3.3.5 Member Data Documentation

3.3.5.1 county

```
string Location::county  [private]
```

Definition at line 135 of file Location.h.

3.3.5.2 latitude

```
float Location::latitude  [private]
```

Definition at line 137 of file Location.h.

3.3.5.3 longitude

```
float Location::longitude  [private]
```

Definition at line 138 of file Location.h.

3.3.5.4 name

```
string Location::name  [private]
```

Definition at line 134 of file Location.h.

3.3.5.5 state

```
string Location::state  [private]
```

Definition at line 136 of file Location.h.

3.3.5.6 zipcode

```
string Location::zipcode  [private]
```

Definition at line 133 of file Location.h.

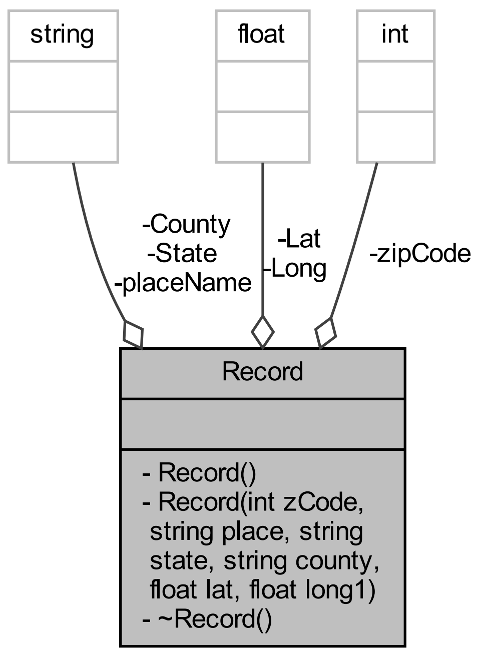
The documentation for this class was generated from the following files:

- Location.h
- Location.cpp

3.4 Record Class Reference

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

Collaboration diagram for Record:


string
float
int
-County
-State
-placeName
-Lat
-Long
-zipCode
Record
- Record()
- Record(int zCode, string place, string state, string county, float lat, float long1)
- ~Record()

Private Member Functions

- Record ()
- Record (int zCode, string place, string state, string county, float lat, float long1)
- ∼Record ()

Private Attributes

- int zipCode = 0
- string placeName = " "
- string State = " "
- string County = " "
- float Lat = 0
- float Long = 0

3.4.1 Detailed Description

Definition at line 17 of file Record.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Record() [1/2]

```
Record::Record ( )  [inline], [private]
```

Definition at line 26 of file Record.h.

3.4.2.2 Record() [2/2]

```
Record::Record (
             int zCode,
             string place,
             string state,
             string county,
             float lat,
             float long1 )  [inline], [private]
```

Definition at line 27 of file Record.h.

3.4.2.3 ∼Record()

```
Record::∼Record ( )  [inline], [private]
```

Definition at line 36 of file Record.h.

3.4.3 Member Data Documentation

3.4.3.1 County

```
string Record::County = " "  [private]
```

Definition at line 22 of file Record.h.

3.4.3.2 Lat

```
float Record::Lat = 0  [private]
```

Definition at line 23 of file Record.h.

3.4.3.3 Long

```
float Record::Long = 0  [private]
```

Definition at line 24 of file Record.h.

3.4.3.4 placeName

```
string Record::placeName = " "  [private]
```

Definition at line 20 of file Record.h.

3.4.3.5 State

```
string Record::State = " "  [private]
```

Definition at line 21 of file Record.h.

3.4.3.6 zipCode

```
int Record::zipCode = 0  [private]
```

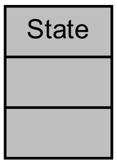
Definition at line 19 of file Record.h.

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

- Record.h

3.5 State Class Reference

Collaboration diagram for State:



3.5.1 Detailed Description

Definition at line 5 of file KeyNode.cpp.

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

- KeyNode.cpp

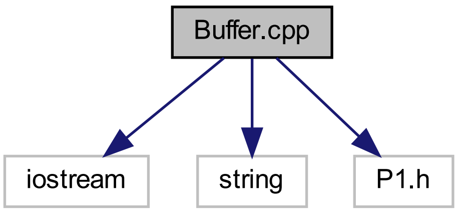
Chapter 4

File Documentation

4.1 Buffer.cpp File Reference

```
#include <iostream>
#include <string>
#include "P1.h"
```

Include dependency graph for Buffer.cpp:



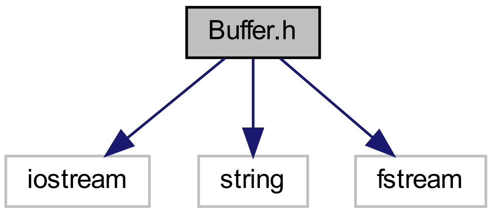
4.2 Buffer.cpp

```
#include <iostream>
#include <string>
//#include "libxl.h"

#include "P1.h"

using namespace std;

int DelimTextBuffer::Read(ifstream & infile)
{
    string info;

    int counter =0;

    infile.open("us_postal_codes.csv");



    if (infile.is_open())
    {
        cout « "File has been opened" « endl;
    }
    else
    {
        cout « "NO FILE HAS BEEN OPENED" « endl;
    }

    while (!infile.eof())
    {
        counter++;
        //infile » info;
        getline(infile, info);
        cout « info « endl;
    }
    infile.close();
    return 0;
}

```

4.3 Buffer.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
```

Include dependency graph for Buffer.h:



Classes

- class DelimTextBuffer

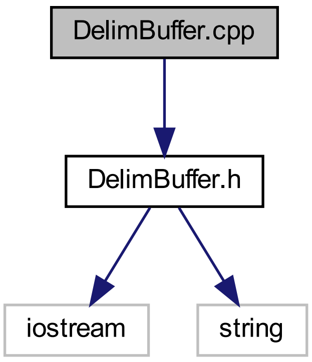
4.4 Buffer.h

```
#include <iostream>
#include <string>
#include <fstream>


using namespace std;

class DelimTextBuffer
{
public:
    DelimTextBuffer(char Delim = ',', int maxBytes = 1000);
    int Read (istream & infile);
    //int Write(ostream & file);
    //int Pack (const char * str, int size = -1);
    //int Unpack (char * str);

private:

    char Delim; // delimiter
    char* Buffer; // Array to hold information piece in
    int Buffersize; // size of field
    int MaxBytes; // Max number of chars in the buffer
    int NextByte; // packing/unpacking position in buffer
};

```

4.5 DelimBuffer.cpp File Reference

`#include "DelimBuffer.h"`

Include dependency graph for DelimBuffer.cpp:



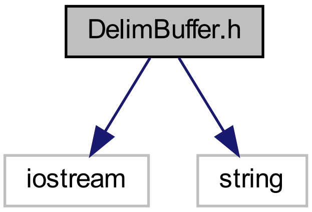
4.6 DelimBuffer.cpp

```


#include "DelimBuffer.h"

void DelimBuffer :: clear()
{
    bufferSize = 0;
    nextCharIndex = 0;
}

bool DelimBuffer :: read (istream& infile)
{
    if (infile.fail()) return false;
    getline(infile, buffer, (char)infile.eof());
    bufferSize = buffer.length();
    return true;
}

bool DelimBuffer :: unpack(string& aStr)
{
    if (nextCharIndex > bufferSize) return false;

    int len = -1; // length of unpacking string
    int start = nextCharIndex; // first character to be unpacked

    for (int i = start; i < bufferSize; i++)
        if (buffer[i] == delim || buffer[i] == '\n') {len = i - start; break;}

    if (len == -1) return false; // delimeter not found
    nextCharIndex += len + 1;

    aStr = buffer.substr(start, len);
    aStr[len] = '\0';
    return true;
}
```

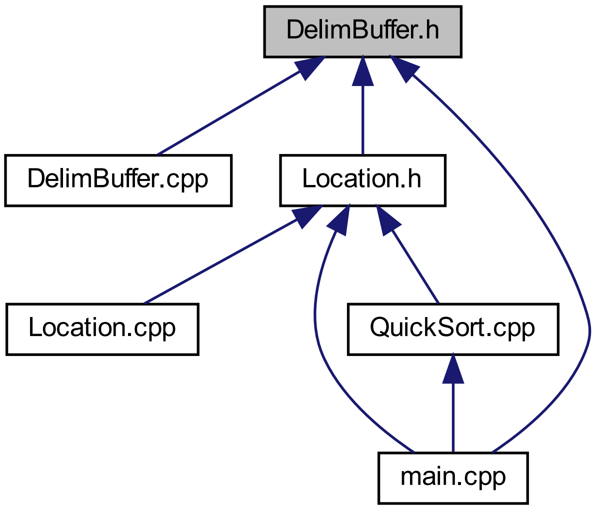
4.7 DelimBuffer.h File Reference

`#include <iostream>`
`#include <string>`

Include dependency graph for DelimBuffer.h:



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



Classes

- class DelimBuffer

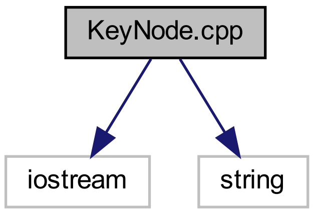
4.8 DelimBuffer.h

```

#ifndef DELIMBUFFER_HEADER
#define DELIMBUFFER_HEADER

#include <iostream>
#include <string>


using namespace std;

class DelimBuffer
{
    public:

        DelimBuffer(char del = ',')
        {
            delim = del;
            bufferSize = 0;
            nextCharIndex = 0;
        }

        bool read(istream& infile); // read stream method
        bool unpack(string& aStr); // unpack the string method

    protected:
        void clear();

    private:
        char delim = ' ';
        string buffer;
        int bufferSize = 0;
        int nextCharIndex = 0; //index of next char
};
#endif
```

4.9 KeyNode.cpp File Reference

```
#include <iostream>
#include <string>
```

Include dependency graph for KeyNode.cpp:



Classes

- class State

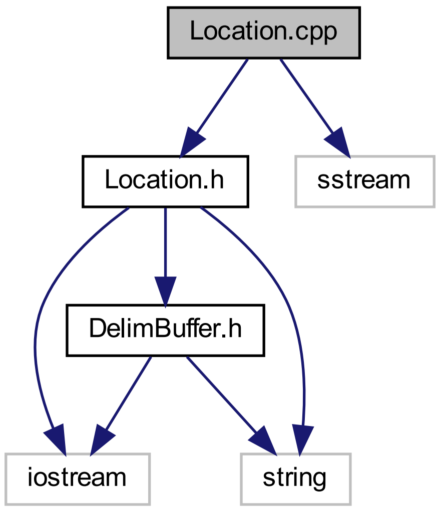
4.10 KeyNode.cpp

```
#include <iostream>
#include <string>
using namespace std;

class State
{

};
```

4.11 Location.cpp File Reference

```
#include "Location.h"
#include <sstream>
```

Include dependency graph for Location.cpp:



Functions

- ostream & operator<< (ostream &out, const Location &loc)

  outstream operator overloaded

4.11.1 Function Documentation

4.11.1.1 operator<<()

```
ostream& operator<< (
            ostream & out,
            const Location & loc )
```

Parameters

outstream	
Location	class object

Returns

returns the overloaded operator right hand operand

Definition at line 81 of file Location.cpp.

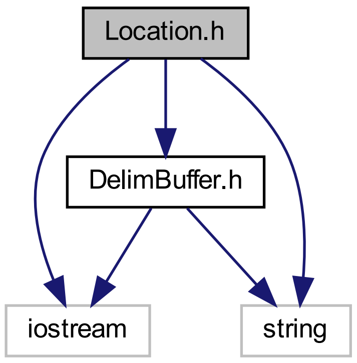
4.12 Location.cpp

```
00001 
00010 
00011 #include "Location.h"
00012 #include <sstream>
00013 
00018 bool Location :: unpack(DelimBuffer &buffer)
00019 {
00020     bool result;
00021     result = buffer.unpack(zipcode);
00022     result = result && buffer.unpack(name);
00023     result = result && buffer.unpack(state);
00024     result = result && buffer.unpack(county);
00025     string lat_str, long_str;
00026     result = result && buffer.unpack(lat_str);
00027     result = result && buffer.unpack(long_str);
00028     stringstream(lat_str) » latitude;
00029     stringstream(long_str) » longitude;
00030     return result;
00031 }
00032 
00033 string Location :: getZipCode() const { return zipcode; } // @return Fetches zipcode value
00034 string Location :: getName() const { return name; } // @return Fetches Name value
00035 string Location :: getCounty() const { return county; } // @return Fetches County value
00036 string Location :: getState() const { return state; } // @return Fetches State value
00037 float Location :: getLat() const { return latitude; } // @return Fetches Latitude value
00038 float Location :: getLong() const { return longitude; } // @return Fetches longitude value
00039 
00042 void Location :: setZipCode(string val) { zipcode = val; }
00043 
00046 void Location :: setName(string val) { name = val; }
00047 
00050 void Location :: setCounty(string val) { county = val; }
00051 
00054 void Location :: setState(string val) { state = val; }
00055 
00058 void Location :: setLat(float val) { latitude = val; }
00059 
00062 void Location :: setLong(float val) { longitude = val; }
00063 
00067 void Location :: operator= (const Location &loc)
00068 {
00069     setZipCode(loc.getZipCode());
00070     setName(loc.getName());
00071     setState(loc.getState());
00072     setCounty(loc.getCounty());
00073     setLat(loc.getLat());
00074     setLong(loc.getLong());
00075 }
00076 
00081 ostream& operator« (ostream& out, const Location &loc)
00082 {
00083     out « loc.getZipCode() « ' '
00084         « loc.getName() « ' '
00085         « loc.getState() « ' '
00086         « loc.getCounty() « ' '
00087         « loc.getLat() « ' '
00088         « loc.getLong() « ' ';
00089     return out;
00090 }
00091 
00095 bool  Location :: operator< (const Location &loc) const
00096 {
00097     return getState() < loc.getState();
}

bool  Location :: operator> (const Location &loc) const
{
    return getState() > loc.getState();
}
```

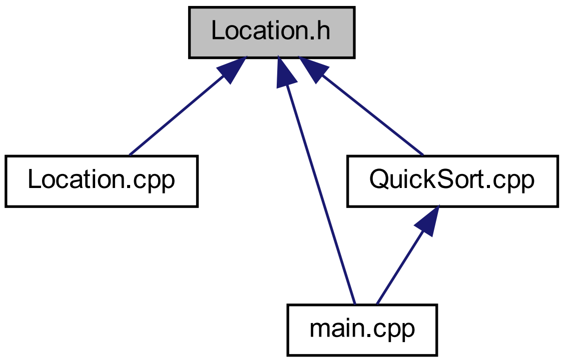
4.13 Location.h File Reference

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

Include dependency graph for Location.h:



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



Classes

- class Location

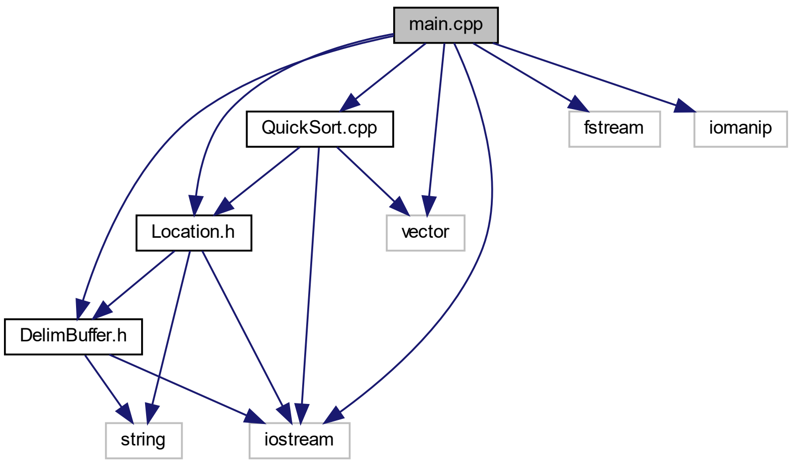
4.14 Location.h

```

#ifndef LOCATION_HEADER
#define LOCATION_HEADER

#include "DelimBuffer.h"
#include <string>
#include <iostream>

using namespace std;

class Location
{
    public:
        Location()
        {
            zipcode = " ";
            name = " ";
            county = " ";
            state = " ";
            latitude = 0;
            longitude = 0;
        };

        Location(string a, string b, string c, string d, float e, float f)
        {
            zipcode = a;
            name = b;
            county = c;
            state = d;
            latitude = e;
            longitude = f;
        };

        Location(const Location& loc)
        {
            zipcode = loc.getZipCode();
            name = loc.getName();
            county = loc.getCounty();
            state = loc.getState();
            latitude = loc.getLat();
            longitude = loc.getLong();
        };

        string getZipCode() const;
        void setZipCode(string val);
        string getName() const;
        void setName(string val);
        string getCounty() const;
        void setCounty(string val);
        string getState() const;
        void setState(string val);
        float getLat() const;
        void setLat(float val);
        float getLong() const;
        void setLong(float val);
        bool unpack(DelimBuffer &buffer);
        void operator= (const Location &loc);
        friend ostream& operator« (ostream& out, const Location &loc);

        bool operator< (const Location &loc) const;

        bool operator> (const Location &loc) const;

    private:
        string zipcode;
        string name;
        string county;
        string state;
        float latitude;
        float longitude;
};
#endif
```

4.15 main.cpp File Reference

```
#include "Location.h"
#include "DelimBuffer.h"
#include <vector>
#include <iostream>
#include <fstream>
#include "QuickSort.cpp"
#include <iomanip>
```

Include dependency graph for main.cpp:



Functions

- int main ()

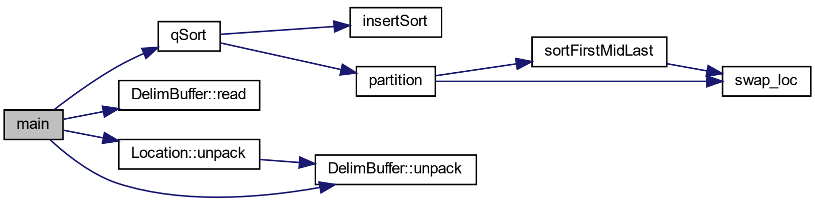
4.15.1 Function Documentation

4.15.1.1 main()

```
int main ( )
```

Definition at line 24 of file main.cpp.

Here is the call graph for this function:



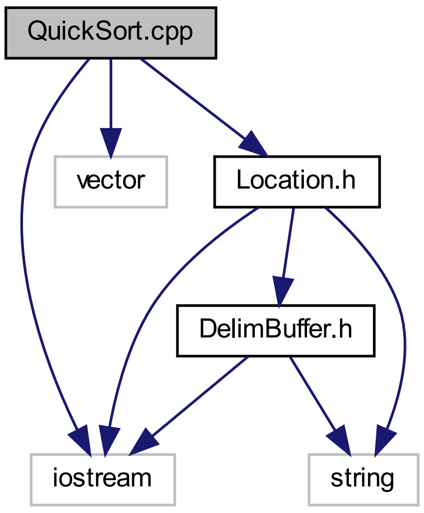
4.16 main.cpp

```


#include "Location.h"
#include "DelimBuffer.h"
#include <vector>
#include <iostream>
#include <fstream>
#include "QuickSort.cpp"
#include <iomanip>


using namespace std;

int main()
{
    vector<Location> location;
    fstream infile;
    infile.open("us_postal_codes.csv");
    DelimBuffer buffer;
    if (!buffer.read(infile)) return 0;
    string field[6];
    for (int i = 0; i < 6; i++) { buffer.unpack(field[i]); }

    while (1)
    {
        Location temp;
        if (!temp.unpack(buffer)) {break;}
        location.push_back(temp);
    }

    int size = location.size();
    qSort(location, 0, size - 1);

    cout « "+-------------------------------------------------------------------+" « endl;
    cout « '|' « setw(5) « "State"
         « '|' « setw(15) « "Westernmost"
         « '|' « setw(15) « "Easternmost"
         « '|' « setw(15) « "Northernmost"
         « '|' « setw(15) « "Southernmost" « '|' « endl;
    cout « "+-------------------------------------------------------------------+" « endl;
    int currentIndex = 0;
    while (currentIndex < size)
    {
        string currentState = location[currentIndex].getState();
        int    w_most = currentIndex,
               e_most = currentIndex,
               s_most = currentIndex,
               n_most = currentIndex;

        currentIndex++;
        while (currentState == location[currentIndex].getState())
        {
            if (location[currentIndex].getLong() > location[w_most].getLong()) w_most = currentIndex;
            if (location[currentIndex].getLong() < location[e_most].getLong()) e_most = currentIndex;
            if (location[currentIndex].getLat() > location[n_most].getLat())   n_most = currentIndex;
            if (location[currentIndex].getLat() < location[s_most].getLat())   s_most = currentIndex;
            currentIndex++;
            if (currentIndex == size) break;
        }
        cout « '|' « setw(5) « location[currentIndex - 1].getState()
             « '|' « setw(15) « location[w_most].getZipCode()
             « '|' « setw(15) « location[e_most].getZipCode()
             « '|' « setw(15) « location[n_most].getZipCode()
             « '|' « setw(15) « location[s_most].getZipCode()  « '|' « endl;
    }
    cout « "+------------------------------------------------------------------------+" « endl;
}

```

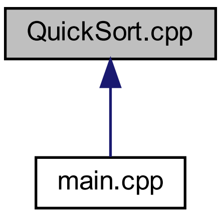
4.17 QuickSort.cpp File Reference

```
#include <iostream>
#include <vector>
#include "Location.h"
```

Include dependency graph for QuickSort.cpp:



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



Functions

- template< class T >
  void swap_loc (vector< T > &aVector, const int &i, const int &j)
- template< class T >
  void insertSort (vector< T > &anArray, const int &first, const int &last)
- template< class T >
  void sortFirstMidLast (vector< T > &anArray, const int &first, const int &mid, const int &last)
- template< class T >
  int partition (vector< T > &anArray, const int &first, const int &last)
- void qSort (vector< Location > &anArray, const int &first, const int &last)

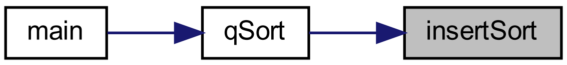
4.17.1 Function Documentation

4.17.1.1 insertSort()

```
template<class T >
void insertSort (
            vector< T > & anArray,
            const int & first,
            const int & last )
```

Definition at line 28 of file QuickSort.cpp.

Here is the caller graph for this function:

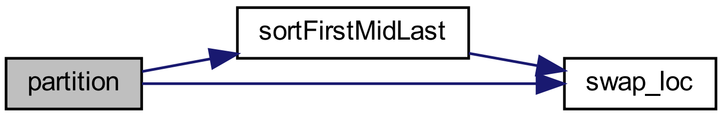


4.17.1.2 partition()

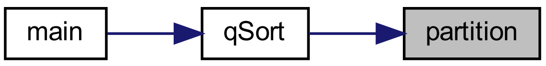
```
template<class T >
int partition (
            vector< T > & anArray,
            const int & first,
            const int & last )
```

Definition at line 75 of file QuickSort.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

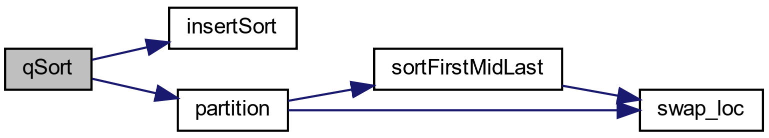


4.17.1.3 qSort()

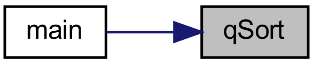
```
void qSort (
             vector< Location > & anArray,
             const int & first,
             const int & last )  [inline]
```

Definition at line 128 of file QuickSort.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

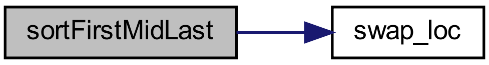


4.17.1.4 sortFirstMidLast()

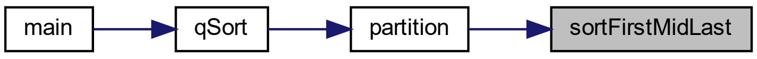
```
template<class T >
void sortFirstMidLast (
             vector< T > & anArray,
             const int & first,
             const int & mid,
             const int & last )
```

Definition at line 62 of file QuickSort.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

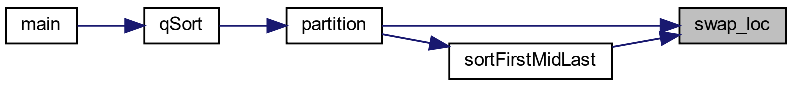


4.17.1.5 swap_loc()

```
template<class T >
void swap_loc (
            vector< T > & aVector,
            const int & i,
            const int & j )
```

Definition at line 20 of file QuickSort.cpp.

Here is the caller graph for this function:



4.18 QuickSort.cpp

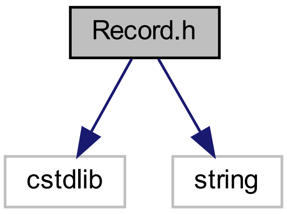
```
00001
00011
00012
00013 #include <iostream>
00014 #include <vector>
00015 #include "Location.h"
00016
00017 using namespace std;
00018
00019 template <class T>
00020 void swap_loc(vector<T> &aVector, const int &i, const int &j)
00021 {
00022     T temp = aVector[i];
00023     aVector[i] = aVector[j];
00024     aVector[j] = temp;
00025 }
00026
00027 template <class T>
00028 void insertSort (vector<T> &anArray, const int &first, const int &last)
00029 {
00030     // unsorted = first index of the unsorted region
00031     // loc = last index the sorted region + 1
00032     // nextItem = next item in the sorted region (the current item that is being placed to the sorted
       region)
00033     for (int unsorted = first + 1; unsorted <= last; unsorted++)
00034     {
00035         // sorted region is anArray[0..unsorted-1]
00036
00037         // get the next item
00038         T nextItem = anArray[unsorted];
00039         // get the right most index of sorted region + 1 (increase size by 1)
00040         int loc = unsorted;
00041
00042         // shift right to make room for the nextItem
00043         while ((loc > 0) && anArray[loc - 1] > nextItem)
00044         {
00045             anArray[loc] = anArray[loc - 1];
00046             loc--;
00047         }
00048
00049         // place the next item to the correct position
00050         anArray[loc] = nextItem;
00051     }
00052 }
00053
00054 /* This function sort 3 first, mid, last entries in increasing order
00055     @pre: first <= mid <= last.
00056     @post: first, mid, last entries are sorted in increasing order
00057     @param: anArray - a given array
00058             first: first index of the first half
```

```
            mid: last index of the first half
            last: last index of the second half */
template <class T>
void sortFirstMidLast (vector<T> &anArray, const int &first, const int &mid, const int &last)
{
    if (anArray[first] > anArray[mid])
    { swap_loc(anArray, first, mid); }

    if (anArray[mid] > anArray[last])
    { swap_loc(anArray, mid, last); }

    if (anArray[first] > anArray[mid])
    { swap_loc(anArray, first, mid); }
}

template <class T>
int partition (vector<T> &anArray, const int &first, const int &last)
{
    // get the middle index
    int mid = first + (last - first)/2;
    // sort first, mid, last
    sortFirstMidLast(anArray, first, mid, last);

    // swap the middle index with the last - 1
    swap_loc(anArray, mid, last - 1);

    // make that pivot
    int pivotIndex = last - 1;
    T pivot = anArray[pivotIndex];

    // start checking from left and right
    int indLeft = first + 1;
    int indRight = last - 2;
    bool done = false;
    while (!done)
    {
        // look for the larger than pivot in the left
        while (anArray[indLeft] < pivot)
        {
            indLeft++;
        }

        // look for smaller than pivot in the right
        while (anArray[indRight] > pivot)
        {
            indRight--;
        }

        // swap them if they are in the wrong side
        if (indLeft < indRight)
        {
            swap_loc(anArray, indLeft, indRight);
            indLeft++;
            indRight--;
        }

        // done with the current pivot
        else
        {
            done = true;
        }
    }

    // swap again to place the pivot to the correct position
    swap_loc(anArray, pivotIndex, indLeft);

    return indLeft;
}

inline void qSort(vector<Location> &anArray, const int &first, const int &last)
{
    if (last - first > 0)
    {
        // if the array size is less than 4, use insert sort
        if ((last - first + 1) < 4)
        {
            insertSort(anArray, first, last);
        }
        // quick sort here
        else
        {
            // find the pivot index
            int pivotIndex = partition(anArray, first, last);

            // divide the array and use quick sort for the 2 subarrays
            qSort(anArray, first, pivotIndex - 1);
            qSort(anArray, pivotIndex + 1, last);
        }
    }
}
```

4.19 Record.h File Reference

`#include <cstdlib>`
`#include <string>`

Include dependency graph for Record.h:



Classes

- class Record

4.20 Record.h

```

//#include <cstring>
#include <cstdlib>
#include <string>

using namespace std;

class Record
{
    int zipCode = 0;
    string placeName = " ";
    string State = " ";
    string County = " ";
    float Lat = 0;
    float Long = 0;

    Record(){};
    Record(int zCode, string place, string state, string county, float lat, float long1)
    {
        zipCode = zCode;
        placeName = place;
        State = state;
        County = county;
        Lat = lat;
        Long = long1;
    }
    ~Record(){};


};
```

Index