SequenceSetGroupProject

v1

Generated by Doxygen 1.8.17

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 SequenceSet::Block Struct Reference

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

Collaboration diagram for SequenceSet::Block:

Public Attributes

- Block ∗ next
- Block ∗ previous
- int records_count
- std::vector< std::string > data

3.1.1 Detailed Description

Definition at line 100 of file SequenceSet.h.

3.1.2 Member Data Documentation

3.1.2.1 next `Block* SequenceSet::Block::next`

Definition at line 101 of file SequenceSet.h.

3.1.2.2 previous `Block * SequenceSet::Block::previous`

Definition at line 101 of file SequenceSet.h.

3.1.2.3 records_count `int SequenceSet::Block::records_count`

Definition at line 102 of file SequenceSet.h.

3.1.2.4 data `std::vector< std::string > SequenceSet::Block::data`

Definition at line 104 of file SequenceSet.h.

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

- SequenceSet.h

3.2 SequenceSet::Index Struct Reference

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

Collaboration diagram for SequenceSet::Index:

Public Attributes

- int key [4]
- Block ∗ block [4]
- Index ∗ subTree [4]
- Index ∗ nextNode
- Index ∗ parent

3.2.1 Detailed Description

Definition at line 110 of file SequenceSet.h.

3.2.2 Member Data Documentation

3.2.2.1 key `int SequenceSet::Index::key[4]`

Definition at line 111 of file SequenceSet.h.

3.2.2.2 block `Block* SequenceSet::Index::block[4]`

Definition at line 112 of file SequenceSet.h.

3.2.2.3 subTree `Index* SequenceSet::Index::subTree[4]`

Definition at line 113 of file SequenceSet.h.

3.2.2.4 nextNode `Index * SequenceSet::Index::nextNode`

Definition at line 113 of file SequenceSet.h.

3.2.2.5 parent `Index * SequenceSet::Index::parent`

Definition at line 113 of file SequenceSet.h.

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

- SequenceSet.h

3.3 SequenceSet Class Reference

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

Collaboration diagram for SequenceSet:

Classes

- struct Block
- struct Index

Public Member Functions

- SequenceSet ()
- SequenceSet (int b_size, int r_size, float d_cap, std::string i_filename, std::string o_filename)

    default constructor.
- ∼SequenceSet ()

    copy constructor
- void create ()

    destructor
- void load ()

    Method: load param:nreturn:npurpose:here we load blocks from the sequence set file into ram.
- void close ()

    Method: close param:none return:none purpose:close files if needed.
- bool is_empty (int flag, int block, int record, int field)
- std::vector< int > search (std::string search_term)
- std::string get_field_from_record (int field, int record, int block)
- void populate ()

    Method: populate param: return: purpose: we count blocks and records.
- void insert (std::string new_record)

    Method: insert param: return: purpose:
- void delete_record (int block, int record)

    Method: remove param: return: purpose:
- void update (int block, int record, int field, std::string new_field)

    Method: update param:int block, int record, int field, std::string new_field return: updated record purpose:update a record.
- void display_record (int record, int block)

    Method: display_record param:int record, int block return: record purpose:display record in a file.
- void display_field (int field, int record, int block)

    Method: display_field param:int field, int record int block return:field in a record purpose:display fields.
- void display_file (int limit)

    Method: display_file param:int limit return:file purpose:return file.
- void display_SS ()

    Method: display_SS param:n/a return:Sequence set purpose:display sequence set.
- void validate ()

    Method: validate param:nreturn:npurpose:validate a record.
- void developer_show ()

    Method: developer_show param:nreturn:npurpose:
- int search_file (int primKey)
- std::vector< int > get_field_range_tuple (int field_index)
- void nsew_most (std::string state)
- void state_and_place_from_zip (std::string zip)

3.3.1 Detailed Description

Definition at line 33 of file SequenceSet.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 SequenceSet() `[1/2]` `SequenceSet::SequenceSet ( )`

\Here we have the first constructor for the SequenceSet i think this will be deleted in the end @param int b_size, int r_size @return n/a @purpose this will initialize some of our data and open the file to default

Definition at line 77 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.2.2 SequenceSet() `[2/2]`

```
SequenceSet::SequenceSet (
        int b_size,
        int r_size,
        float d_cap,
        std::string i_filename,
        std::string o_filename )
```

default constructor.

\Here we have the constructor for the SequenceSet that takes in all the values relivant to the header and saving @param int b_size, int r_size, int d_cap, std::string i_filename, std::string o_filename @return n/a @purpose this will initialize some of our data and open the file and output file \this is the constructor for the header

Definition at line 98 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.2.3 ∼SequenceSet() `SequenceSet::∼SequenceSet ( )`

copy constructor

\Here we have the first destructor for the SequenceSet @param n/a @return n/a @purpose this free memory

Definition at line 118 of file SequenceSet.cpp.

3.3.3 Member Function Documentation

3.3.3.1 create() `void SequenceSet::create ( )`

destructor

Method: create param:none return:none purpose: this will create the empty file with just the header and any data in the data array.

Your header record should include the following components: –sequence set file type –header record size –block size {default to (512B / block)} –maximum count of records per block –minimum capacity: 50% -(for simplicity, require an even number) –record size –count of fields per record –field info triple (tuple) {AoS or SoA} –name or ID –size –type schema -(format to read or write) -indicate field which serves as the primary key –pointer to the block avail-list -pointer to the active sequence set list -block count -record count -stale flag -Simple Index (10.3) -file name -schema information here i am making the header components to be at the top of the file

here is a disign desicion: SoA or AoS here structure

here is an array of structures

write the header

Definition at line 155 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.2 load() `void SequenceSet::load ( )`

Method: load param:n*return:npurpose:here we* load blocks from the sequence set file into ram.

function prototype for create() that creates empty file for the header any it contains create a local file for loading in that data

if the file ended then tell the user and exit

go through each line of the file

if we find the end of header tag then break

cut it into words. look for "Fields:" Record the feilds and stop

here are the strings to find what field is in what spot. a function to strip spaces would be ideal here

here are the store of index's for what in what order

get the line

split it into section

for each one see if it is one of the identifiers above and if so store its location

start a counter

go while we still have lines and are not taking too many fields

if we find the end of header tag then break

take each line which will house the field data

chop it and put it into the correct vector to be used later.

increase since we have another field that was specified

close files

Definition at line 219 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.3 close() `void SequenceSet::close ( )`

Method: close param:none return:none purpose:close files if needed.

function prototype for load() that load block of sequence set file into ram

Definition at line 301 of file SequenceSet.cpp.

3.3.3.4 is_empty()

```
bool SequenceSet::is_empty (
            int flag,
            int block = -1,
            int record = -1,
            int field = -1 )
```

function prototype for close() that is called when file needs to be closed

Method: is_open param: int flag 0 - file 1 - block 2 - record 3 - field return: bool true if empty and false if populated purpose: to know the state of a structure

if nothing is given but the flag then it will do input/output on command line for user and will take in the index's of requested whatever structure and tell you its status this will check the status of requested

file or the whole linked list

block

record or field

block is valid

Definition at line 334 of file SequenceSet.cpp.

3.3.3.5 search()

```
std::vector< int > SequenceSet::search (
            std::string search_term )
```

prototype for is_empty() to know the state of the structure

Method: search param: return: purpose:

Definition at line 393 of file SequenceSet.cpp.

3.3.3.6 get_field_from_record()

```
std::string SequenceSet::get_field_from_record (
            int field,
            int record,
            int block )
```

function prototype for search(string) to search for specific record in the file from user input if we are not in range or acceptable give null

Definition at line 762 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.7 populate() `void SequenceSet::populate ( )`

Method: populate param: return: purpose: we count blocks and records.

we open the file and skip the header and loop the rest we create an empty node for a btree current record being coppied

current block number int primary_key_i;

pointer to current node in_file.open(in_filename);

make btree node as neccessary

make a new node every 3 primary keys

first node skips this

move onto next node

remember where we are

node count is 0

if first then its the root

fill the children of the b tree

make next (or first) Block:

get the empty block

prev is null

next too

resize the array to be the length of the block sizes

resize it for the length of a record //ERROR

increase block count each iteration, and if it isnt 0 like the first iteration then set the first in the sequence set to be b

if not then send it to the next node.

while block isn't __% full, keep filling:

get the primary key and add it to the tree DO CONTINUE ON FROM HERE

Build the B+ tree up from the "linked list" structure

Definition at line 433 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.8 insert()

```
void SequenceSet::insert (
    std::string new_record )
```

Method: insert param: return: purpose:

function prototype for populate() that creates an empty node for a btree all blocks filled make a new one

Definition at line 530 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.9 delete_record()

```
void SequenceSet::delete_record (
    int block = -1,
    int record = -1 )
```

Method: remove param: return: purpose:

function prototype for insert(strint) that inserts a new record into the file from user input

Definition at line 597 of file SequenceSet.cpp.

3.3.3.10 update()

```
void SequenceSet::update (
    int block,
    int record,
    int field,
    std::string new_field )
```

Method: update param:int block, int record, int field, std::string new_field return: updated record purpose:update a record.

function prototype for delete_record(int, int) that deletes specific record from user input

Definition at line 630 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.11 display_record()

```
void SequenceSet::display_record (
    int record = -1,
    int block = -1 )
```

Method: display_record param:int record, int block return: record purpose:display record in a file.

function prototype for update(int, int, string) that updates a record, field or adds new field

Definition at line 675 of file SequenceSet.cpp.

3.3.3.12 display_field() `void SequenceSet::display_field (`

```
int field = -1,
int record = -1,
int block = -1 )
```

Method: display_field param:int field, int record int block return:field in a record purpose:display fields.

function prototype display_record(int, int) displays specific record request by user input

Definition at line 714 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.13 display_file() `void SequenceSet::display_file (`

```
int limit = -1 )
```

Method: display_file param:int limit return:file purpose:return file.

function prototype display_field(int, int, int) displays specific field request by user input

Definition at line 800 of file SequenceSet.cpp.

3.3.3.14 display_SS() `void SequenceSet::display_SS ( )`

Method: display_SS param:n/a return:Sequence set purpose:display sequence set.

function prototype display_file(int) displays file request by user input

Definition at line 824 of file SequenceSet.cpp.

3.3.3.15 validate() `void SequenceSet::validate ( )`

Method: validate param:n*return:npurpose:validate a* record.

function prototype display_SS() to display the sequence set

Definition at line 866 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.16 developer_show() `void SequenceSet::developer_show ( )`

Method: developer_show param:n*return:npurpose:*

function prototype validate() to validate a record in the file

Definition at line 899 of file SequenceSet.cpp.

3.3.3.17 search_file() `int SequenceSet::search_file ( int primKey )`

function prototype developer_show() that creates the columns the record will be diplayed into

3.3.3.18 get_field_range_tuple() `std::vector< int > SequenceSet::get_field_range_tuple ( int field_index )`

function prototype search_file(int) searches for a file

often we want where the characters index is in the recod which is stored in the range but extracting that range isnt easy so heres a function to do it

Definition at line 916 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.19 nsew_most() `void SequenceSet::nsew_most ( std::string state )`

function prototype get_field_range_tuple(int) for extracting the range of character index in a record

Definition at line 968 of file SequenceSet.cpp.

Here is the call graph for this function:

3.3.3.20 state_and_place_from_zip() `void SequenceSet::state_and_place_from_zip ( std::string zip )`

Definition at line 927 of file SequenceSet.cpp.

Here is the call graph for this function:

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

- SequenceSet.h
- SequenceSet.cpp

4 File Documentation

4.1 main.cpp File Reference

```
#include "SequenceSet.cpp"
```

Include dependency graph for main.cpp:

4.2 main.cpp

```
/*
  Authors: Jacob Hopkins, Misky Abshir, and Tyler Willard
  Date: 4/27/2020
  Due: 5/1/2020

  TODO:
    - Create SequenceSet Class in the ’SequenceSet.h’ file
    - Create a test driver for the class above
    - Create the program for using the txt files with the class
    - Create the design doccument for
        - The Class
        - The Test Driver
    - A User Manual
    - DyOxygen the Code

 Specifications for a program which uses the Sequence Set class:
    The application program will iterate through the sequence set displaying (neatly)
    the Northernmost, Southernmost, Easternmost, and Westernmost zip code for a specified state.
    {You can verify the results by sorting the Excel source data file first by state, then by longitude or by latitude}

    Also, the application program, using a different set of command line flags,
    will display (neatly) the State and Place Name for a specified Zip Code (or set of Zip Codes).

    Run the test driver program to build the full Sequence Set file (and index file);
    Run the application program specifying the Sequence Set file and the State on the command line;
    Use the Unix script command to show:
        the building of the Sequence Set file,
        the repeated running of the application program and its output for several states,
        the repeated running the application program to display the State and Place Name for several Zip Codes.


  In Video 4:
  14:30

  I don’t mind phone call for audi as well, 612-707-2182 that’s my cellphone
*/


//import header with the SequenceSet class
#include "SequenceSet.cpp"
#include <cstdio>

/*
  Here is the main function to start the program.
*/
int main(int arg_count, char** arg_values){

  /*
    Here we declare and initialize the sequence set data. This will call load().
  */

  SequenceSet data;

  data.create();

  data.populate();

  //while(true)
  //  data.display_field();

  //data.display_record(); //works most of the time, with the occasional exit

  //data.display_file();          //this works great

  //data.display_SS();          //this works great

  //data.developer_show();      //this works great

  /*
  std::vector<int> loc = data.search(data.get_field_from_record(0,0,0));
  std::cout « "\n" « data.get_field_from_record(0,0,0) « "\nBlock:\t" « std::to_string(loc[0]) « "\nRecord:\t" « std::to_string(loc[1]) « "\n";
  loc = data.search(data.get_field_from_record(4,1,0));
  std::cout « "\n" « data.get_field_from_record(4,1,0) « "\nBlock:\t" « std::to_string(loc[0]) « "\nRecord:\t" « std::to_string(loc[1]) « "\n";
  loc = data.search(data.get_field_from_record(4,1,1));
  std::cout « "\n" « data.get_field_from_record(4,1,1) « "\nBlock:\t" « std::to_string(loc[0]) « "\nRecord:\t" « std::to_string(loc[1]) « "\n";
  loc = data.search("42.1934"); // from line 28
  std::cout « "\n42.1934" « "\nBlock:\t" « std::to_string(loc[0]) « "\nRecord:\t" « std::to_string(loc[1]) « "\n";
  loc = data.search("yeeeet"); // from line 28
  std::cout « "\nyeeeet" « "\nBlock:\t" « std::to_string(loc[0]) « "\nRecord:\t" «
```

```
    std::to_string(loc[1]) « "\n";
 */

 //data.insert("");

 /*
 data.update(0,0,0,"12345");
 data.update(0,0,1,"12345");
 data.update(0,0,2,"12345");
 data.update(0,0,3,"12345");
 data.update(0,0,4," 12.345");
 data.update(0,0,5,"-12.345");
 */


 //data.display_SS();
 //data.validate();



 //std::cout « "You have entered " « arg_count « " arguments:" « "\n";
 //for (int i = 0; i < arg_count; ++i)
 //    std::cout « arg_values[i] « "\n";

 if(*arg_values[1] == 'a'){
   std::cout « "Finding the furthest zip codes in: " « arg_values[2] « "\n";
   data.nsew_most(arg_values[2]);
 }

 if(*arg_values[1] == 'b'){
   for(int i = 2; i < arg_count; i++){
     std::cout « "Finding the State and Place name of zip code: " « arg_values[i] « "\n";
     data.state_and_place_from_zip(arg_values[i]);
   }
 }

 //forbiden code here
 //wait for character so the screen does not disappear
 std::cout « "Press enter...";
 getchar();

 //return that the program ran correctly
 return 0;
}
```

4.3 readme.txt File Reference

4.4 scipt_test.txt File Reference

4.5 script_main.txt File Reference

Variables

- Script started on

4.5.1 Variable Documentation

4.5.1.1 on `Script started on`

Definition at line 8 of file script_main.txt.

4.6 SequenceSet.cpp File Reference

```
#include <iostream>
#include <cstdio>
#include <fstream>
#include "SequenceSet.h"
#include <string>
#include <sstream>
```

Include dependency graph for SequenceSet.cpp: This graph shows which files directly or indirectly include this file:

Functions

- std::vector< std::string > split_string (std::string str, char delimiter)

  Utility Methods.
- std::vector< char > string_to_vector (std::string s, int n)
- std::string add_c_to_a_til_size_of_b (std::string a, std::string b, std::string c, bool front=true)
- std::string add_c_to_a_til_size_of_b (std::string a, int b, std::string c, bool front=true)

4.6.1 Function Documentation

4.6.1.1 split_string()

```
std::vector<std::string> split_string (
        std::string str,
        char delimiter )
```

Utility Methods.

param string str, char delimiter return purpose This function will take apart a string and split it by some char delimeter

Definition at line 15 of file SequenceSet.cpp.

4.6.1.2 string_to_vector()

```
std::vector<char> string_to_vector (
        std::string s,
        int n )
```

Definition at line 33 of file SequenceSet.cpp.

4.6.1.3 add_c_to_a_til_size_of_b() [1/2]

```
std::string add_c_to_a_til_size_of_b (
        std::string a,
        std::string b,
        std::string c,
        bool front = true )
```

Definition at line 42 of file SequenceSet.cpp.

4.6.1.4 add_c_to_a_til_size_of_b() [2/2]

```
std::string add_c_to_a_til_size_of_b (
        std::string a,
        int b,
        std::string c,
        bool front = true )
```

Definition at line 55 of file SequenceSet.cpp.

4.7 SequenceSet.cpp

```
00001 #include <iostream>
00002 #include <cstdio>
00003 #include <fstream>
00004 #include "SequenceSet.h"
00005 #include <string>
00006 #include <sstream>
00007
00009
00015 std::vector<std::string> split_string(std::string str, char delimiter){
00016   std::vector<std::string> split_str;
00017   std::string word;
00018
00019   for(char x : str){
00020
00021     if (x == delimiter){
00022       split_str.push_back(word);
00023       word = "";
00024     }else{
00025       word = word + x;
00026     }
00027
00028   }
00029   split_str.push_back(word);
00030   return split_str;
00031 }
00032
00033 std::vector<char> string_to_vector(std::string s, int n){
00034   std::vector<char> v;
00035   v.push_back(n);
00036   for (char c : s){
00037     v.push_back(c);
00038   }
00039   return v;
00040 }
00041
00042 std::string add_c_to_a_til_size_of_b(std::string a,std::string b,std::string c, bool front = true){
00043   int size_to_be = b.size();
00044   std::string new_a = a;
00045   while(new_a.size() < size_to_be){
00046     if(front){
00047       new_a = c + new_a;
00048     }else{
00049       new_a = new_a + c;
00050     }
00051   }
00052   return new_a;
00053 }
00054
00055 std::string add_c_to_a_til_size_of_b(std::string a,int b,std::string c, bool front = true){
00056   std::string new_a = a;
00057   while(new_a.size() < b){
00058     if(front){
00059       new_a = c + new_a;
00060     }else{
00061       new_a = new_a + c;
00062     }
00063   }
00064   return new_a;
00065 }
00066
00067
00068 /*   Class Methods   */
00069
00077 SequenceSet::SequenceSet(){
00078   block_size = 512;  //records per block
00079   record_size = 1; //characters per record
00080   in_filename = "us_postal_codes_formatted.txt";
00081   out_filename = "us_postal_codes_sequence_set_file.txt";
00082   default_cap = 0.5;
00083   primary_key_index = 0;
00084   first = NULL;
```

```
  end_of_header = "12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890";

  load();
}

SequenceSet::SequenceSet(int b_size, int r_size, float d_cap, std::string i_filename, std::string o_filename){
  block_size = b_size;
  record_size = r_size;
  default_cap = d_cap;
  in_filename = i_filename;
  out_filename = o_filename;
  primary_key_index = 0;
  first = NULL;
  end_of_header = "12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890";

  load();
}

SequenceSet::~SequenceSet(){
  delete(&field_count, &block_size, &record_size, &default_cap, &in_filename, &out_filename, &first, &root, &primary_key_index);
  delete(&end_of_header, &in_file, &out_file, &field_labels, &field_sizes, &field_types);
}


void SequenceSet::create(){
  std::string file_type = "ascii";
  std::string header_record_size = "22 lines";
  record_size = -1;
  int max_record_count = -1;
  int f_count = field_count;
  Block* block_avail = first;
  Index* active_list = root;
  int block_count = 0;
  int record_count = 0;
  bool stale = false;

  struct field_tuple{
    std::string label;
    std::string size;
    std::string type;

    field_tuple(std::string a,std::string b,std::string c){
      label = a;
      size = b;
      type = c;
    };
  };
  std::vector<field_tuple> fields;
  for (int i = 0; i < field_count; i++){
    fields.push_back(field_tuple(field_labels[i],field_sizes[i],field_types[i]));
  }

  out_file.open(out_filename);

  out_file « "File Type: " « file_type « "\n";
  out_file « "Header Size: " «  header_record_size « "\n";
  out_file « "Block Size: " «  block_size « "\n";
  out_file « "Maximum Records: " « max_record_count  « "\n";
  out_file « "Minimum Capacity: " « default_cap  « "%\n";
  out_file « "Record Size: " «  record_size « "\n";
  out_file « "Record Field Count: " « field_count  « "\n";
  for (field_tuple f : fields){
    out_file « f.label « '|' « f.size « '|' « f.type « "\n";
  }
  out_file « "Primary Key: " « field_labels[0] « "\n";
  out_file « "Avail Block Pointer: " «  block_avail  « "\n";
  out_file « "Active List: " «  active_list « "\n";
  out_file « "Block Count: " «  block_count  « "\n";
  out_file « "Record Count: " «  record_count « "\n";
  out_file « "Stale Flag: " «  stale « "\n";
  out_file « out_filename « "\n";
  out_file « "This file is for loading blocks into a sequence set." « "\n";
  out_file « end_of_header « "\n";

  close();
}


void SequenceSet::load(){
  std::string line = "";
  in_file.open(in_filename);

  if (in_file.fail()) {
    exit(1);
  }

  while(std::getline(in_file, line)){
    if(!end_of_header.compare(line))
      break;

    std::vector<std::string> spaceless_line = split_string(line, ' ');
    if (!spaceless_line[0].compare("Fields:")){
      std::stringstream field_count_string(spaceless_line[1]);
      field_count_string » field_count;
      break;
    }
  }

  std::string field_name_identifier = "Field Name  ";
  std::string column_range_identifier = "   column range    ";
  std::string type_identifier = "   type ";

  int index_of_field_name = -1;
  int index_of_collum_size = -1;
  int index_of_type = -1;

  std::getline(in_file, line);
  std::vector<std::string> field_data_positions = split_string(line, '|');
  for (int i = 0; i < field_data_positions.size(); i++){
    if (!field_data_positions[i].compare(field_name_identifier))
      index_of_field_name = i;
    if (!field_data_positions[i].compare(column_range_identifier))
      index_of_collum_size = i;
    if (!field_data_positions[i].compare(type_identifier))
      index_of_type = i;
  }


  int i = 0;
  while (std::getline(in_file, line) && i < field_count){
    if(!end_of_header.compare(line))
      break;

    std::vector<std::string> field_data_split = split_string(line, '|');
    field_labels.push_back(field_data_split[index_of_field_name]);
    field_sizes.push_back(field_data_split[index_of_collum_size]);
    field_types.push_back(field_data_split[index_of_type]);

    i++;
  }

  close();
}


void SequenceSet::close(){

  Block *b = first;
  while( b != NULL){

    b = b -> next;
  }
  delete(b);


  if (in_file.is_open()) {
    in_file.close();
  }
  if (out_file.is_open()) {
    out_file.close();
  }
}


bool SequenceSet::is_empty(int flag, int block = -1, int record = -1, int field = -1){
  bool status;

  if(flag == 0){
    status = (first == NULL);
  }

  if(flag == 1){

    if (block == -1){
      std::cout « "Index of Block to check: ";
      std::cin » block;
    }

    Block *b = first;
    while(block > 0){
      status = (b==NULL);
      b = b -> next;
      block--;
    }

  }

  if(flag == 2 || flag == 3){
    if (block == -1){
      std::cout « "Index of Block to check: ";
      std::cin » block;
    }

    Block *b = first;
    while(block > 0){
      status = (b==NULL);
      b = b -> next;
      block--;
    }

    if(!status){
      if (record == -1){
        std::cout « "Index of Record to check: ";
        std::cin » record;
      }

      status = (b -> data[record] == "");

    }
  }

  return status;
}


std::vector<int> SequenceSet::search(std::string search_term){
  Block *b = first;
  std::vector<int> loc;
  int block_count = -1, record_count = -1;

  while(b != NULL){
    block_count++;
    record_count = -1;
    for(std::string record : b -> data){
      record_count++;
      if(record_count > b -> records_count){
        break;
      }
      if(record.size() <= search_term.size()){
        if(record.find(search_term, 0) != std::string::npos){
          loc.push_back(block_count);
          loc.push_back(record_count);
          return loc;
        }
      }
    }

    b = b -> next;
  }
  loc.push_back(-1);
  loc.push_back(-1);
  return loc;
}


void SequenceSet::populate(){
  int record_number;
  int block_count = -1;

  int primary_key_int;
  std::string primary_key_tmp;
  int index_place = -1, node_count = 0;


  Block *prev;
  Index *current_node = new Index;

  std::string line = "";

  in_file.open(in_filename);

  while(std::getline(in_file,line)){
    if(!line.compare(end_of_header)){
      break;
    }
  }

  while(!in_file.eof()){
    record_number = 0;

    if(++index_place%3 == 0){
      if(node_count++ > 0){
        current_node->nextNode = new Index;
        current_node = current_node->nextNode;
      }else
        root = current_node;

      for (int i = 0; i < 4; i++){
        current_node -> key[i] = -1;
        current_node -> block[i] = NULL;
        current_node -> block[i] = NULL;
      }
      current_node -> nextNode = NULL;
      current_node -> parent = NULL;
    }

    Block *b = new Block;
    b -> previous = NULL;
    b -> next = NULL;
    b -> data.resize(block_size);
    for(int i = 0; i < block_size; i++){
      b -> data[i] = "";
    }

    if(++block_count != 0){
      b -> previous = prev;
      prev -> next = b;
    }
    else
      first = b;
    prev = b;


    std::string line;
    while(record_number < (block_size * default_cap) && !in_file.eof()){
      std::getline(in_file, line);
      if(line != " "){
        //std::cout « record_number « "  -" « line « "-\n";
        prev -> data[record_number] = add_c_to_a_til_size_of_b(std::to_string(record_number), std::to_string(block_size), "0") + line;
        record_number++;
      }
    }

    std::string tmp = prev->data[record_number];
    primary_key_tmp = tmp.substr(0,5);
    primary_key_tmp.resize(6);
    primary_key_int = atoi(primary_key_tmp.c_str());
    current_node -> key[index_place%3] = primary_key_int;
    current_node -> block[index_place%3] = prev;

    prev -> records_count = record_number;

  }
  delete(prev);


  close();
}


void SequenceSet::insert(std::string new_record){
  if(new_record == ""){
    bool f = true;
    std::vector<std::string> constructed_record;
    std::vector<int> ranges = {}, ranges_2 = {};
    std::string term;
    int i = 0;
    for(std::string field: field_labels){
      term = "";
      ranges = get_field_range_tuple(i);
      int length = (ranges[1] - (ranges[0]))+1;
      if(i >= 1){
        ranges_2 = get_field_range_tuple(i-1);
        if(ranges[0] - ranges_2[1] >= 2){
          constructed_record.push_back(" ");
        }
      }
      while(term.size() != length){
        std::cout « "Input " « field « ": ";
        std::cin » term;
        term = add_c_to_a_til_size_of_b(term, length," ",f);
      }
      f = false;
      constructed_record.push_back(term);
      i++;
    }
    for(std::string s: constructed_record){
      new_record = new_record + s;
    }
  }

  Block *b = first;
  bool placed = false;
  int block = -1;

  while( b != NULL && !placed){
    block++;
    if(b -> records_count < block_size){
      b -> records_count++;
      b -> data[b -> records_count - 1] = std::to_string(b -> records_count-1) + " " + new_record;
      placed = true;
    }
    b = b -> next;
  }

  if(!placed){
    block++;
    Block *new_b = new Block;
    new_b -> previous = b;
    b -> next = new_b;
    new_b -> records_count++;
    new_b -> data[new_b -> records_count - 1] = std::to_string(new_b -> records_count - 1) + " " + new_record;
  }

  std::cout « "\nInserted into: \nBlock\t" « block « "\nRecord\t" « b -> records_count - 1 « "\n";
  delete(b);
}


void SequenceSet::delete_record(int block = -1, int record = -1){
  Block *b = first;
  int b_count = 0, r_count;

  if(block == -1){
    std::cout « "Enter block index: " « std::endl;
    std::cin » block;
  }
  if(record == -1){
    std::cout « "Enter record index: " « std::endl;
    std::cin » record;
  }

  while( b != NULL && b_count < block){
    b_count++;
    b = b -> next;
  }

  if (record > 0 && record < b -> records_count){
    b -> data[record] = "";
  }

  delete(b);
}


void SequenceSet::update(int block, int record, int field, std::string new_field){
  Block *b = first;
  int b_count = 0;

  while(b != NULL && b_count < block){
    b -> next;
    b_count++;
  }
  bool front = field == 0;
  if(record >=0 && record < block_size){
    if(field >= 0 && field < field_count){
      std::string updated = "", current = b -> data[record];

      std::vector<int> loc = get_field_range_tuple(field);
      int length = (loc[1] - loc[0])+1;
      bool added = false;
      int count = -2;
      for(char c : current){
        if(count < loc[0] || count > loc[1]){
          updated = updated + c;
        }else if(!added){
          added = true;
          if(new_field.size() <= length){
            updated = updated + add_c_to_a_til_size_of_b(new_field, length, " ", front);
          }else{
            updated = updated + new_field.substr(0, length);
          }
        }
        count++;
      }
      std::cout « updated « "\n";
      b -> data[record] = updated;
    }
  }
  delete(b);
}


void SequenceSet::display_record(int record = -1, int block = -1){
  Block *b = first;
  int b_count = 0;

  while(block < 0){
    std::cout « "\nEnter block index: ";
    std::cin » block;
  }
  while(record < 0){
    std::cout « "\nEnter record index: ";
    std::cin » record;
  }

  while( b != NULL && b_count < block){
    b_count++;
    b = b -> next;
  }

  std::string record_s = "*Record Not Found*";
  if(b != NULL){
    int size = b->records_count;
    if(record >= 0 && record < size){
      record_s = b -> data[record];
    }else if(record >= 0 && record <= block_size){
      record_s = "*Empty Record*";
    }
  }

  std::cout « "\n\'" « record_s « "\'\n";
  delete(b);
}

void SequenceSet::display_field(int field = -1, int record = -1, int block = -1){
  Block *b = first;
  int b_count = 0;

  while(block < 0){
    std::cout « "\nEnter block index: ";
    std::cin » block;
  }
  while(record < 0){
    std::cout « "\nEnter record index: ";
    std::cin » record;
  }
  while(field < 0){
    std::cout « "\nEnter field index: ";
    std::cin » field;
  }

  while( b != NULL && b_count < block){
    b_count++;
    b = b -> next;
  }

  std::string record_s = "*Record Not Found*";
  if(b != NULL){
    int size = b->records_count;
    if(record >= 0 && record < size){
      record_s = b -> data[record];

      if(field >= 0 && field < field_count){
        std::vector<int> ranges = get_field_range_tuple(field);
        int length = (ranges[1] - (ranges[0]-1));
        int start = ranges[0]-1 + std::to_string(block_size).size();
        std::string field_s = record_s.substr(start, length);
        //std::cout « "\n\'" « ranges[0] « "-" « ranges[1] « "\'\n";
        std::cout « "\n\'" « field_s « "\'\n";
        return;
      }

    }else if(record >= 0 && record <= block_size){
      record_s = "*Empty Record*";
      std::cout « "\n\'" « record_s « "\'\n";
    }
  }

  std::cout « "\n\'" « record_s « "\'\n";
  delete(b);
}

std::string SequenceSet::get_field_from_record(int field, int record, int block){
  Block *b = first;
  int b_count = 0;

  while( b != NULL && b_count < block){
    b_count++;
    b = b -> next;
  }

  std::string record_s;
  if(b != NULL){
    int size = b->records_count;
    if(record >= 0 && record < size){
      record_s = b -> data[record];

      if(field >= 0 && field < field_count){
        std::vector<int> ranges = get_field_range_tuple(field);
        int length = (ranges[1] - (ranges[0]-1));
        int start = ranges[0]-1 + std::to_string(block_size).size();
        std::string field_s = record_s.substr(start, length);
        return field_s;
      }

    }else if(record >= 0 && record <= block_size){
      return "";
    }
  }
  delete(b);
  return NULL;
}

void SequenceSet::display_file(int limit = -1){
  if(limit == -1){limit = block_size;}
  Block *b = first;
  int count = 0;
  while( b != NULL && count < limit){
    std::cout « "Block " « count « "\n";
    std::cout « "Records in Block " « count « ": " « b -> records_count « "\n";
    std::cout « "Head of Records: \n" « b -> data[0] « "\n";
    int last = b -> records_count - 1;
    std::cout « "Tail of Records: \n" « b -> data[last] « "\n\n";
    count++;
    b = b -> next;
  }
  delete(b);
}


void SequenceSet::display_SS(){
  Block *b = first;
  int count = 0;
  std::string empty_records_index_string;

  std::cout « "\n\nPress enter to see next block...(Ctrl + C to stop)";
  getchar();

  while( b != NULL){
    std::cout « "\nBlock " « count « "\n";
    std::cout « "Records in Block " « count « ": " « b -> records_count « "\n";
    int r_count = 0;
    empty_records_index_string = "[";
    for(std::string r : b -> data){
      if(r != ""){
        std::cout « "Record " « r_count « ": \"" « r « "\"\n";
        r_count++;
      }else{
        empty_records_index_string = empty_records_index_string + std::to_string(r_count) + ", ";
        r_count++;
      }
    }
    std::cout « "Empty Records: " « empty_records_index_string « "]\n";
    std::cout « "\n\n";

    std::cout « "Press enter to see next block...(Ctrl + C to stop)";
    getchar();

    count++;
    b = b -> next;
  }
  delete(b);
```

```
}


void SequenceSet::validate(){
  Block *b = first;
  std::vector<int> loc = get_field_range_tuple(0);
  int start = 1 + std::to_string(block_size).size();
  int length = 1 + ( loc[1] - loc[0] );
  bool error = false;
  while( b != NULL){
    int last = b -> records_count;
    for (int i = 0; i < last-1; i++){
      int prev = atoi(b -> data[i].substr(start, length).c_str());
      int current = atoi(b -> data[i+1].substr(start, length).c_str());
      if( prev > current){
        error = true;
        std::cout « "Out Of Order: " « i « "\n";
      }
    }
    b = b -> next;
  }
  if(!error){
    std::cout « "Validated to be: In Order." « "\n";
  }
  delete(b);
}



void SequenceSet::developer_show(){
  std::cout « "field_count:\t" « field_count « "\n";

  std::cout « "field_labels|field_sizes|field_types \n";
  for (int i = 0; i < field_labels.size(); i++){
    std::cout « field_labels[i] « "|" « field_sizes[i] « "|" « field_types[i] « "\n";
  }
  std::cout « "\n";
}

std::vector<int> SequenceSet::get_field_range_tuple(int field_index){
  std::string s = field_sizes[field_index];
  std::vector<std::string> sub_s = split_string(s, '-');
  int low = atoi(sub_s[0].c_str());
  int high = atoi(sub_s[1].c_str());
  std::vector<int> r = {low,high};
  return r;
}



void SequenceSet::state_and_place_from_zip(std::string zip){
  Block *b = first;
  std::string rec, zip_s, state_s, place_s;

  std::vector<int> loc_zip = get_field_range_tuple(0);
  int start_zip = std::to_string(block_size).size() + loc_zip[0] - 1, length_zip = loc_zip[1] -
 loc_zip[0] + 1;

  std::vector<int> loc = get_field_range_tuple(2);
  int start_state = std::to_string(block_size).size() + loc[0] - 1, length_state = loc[1] - loc[0] +
 1;

  std::vector<int> loc_p = get_field_range_tuple(1);
  int start_place = std::to_string(block_size).size() + loc_p[0] - 1, length_place = loc_p[1] -
 loc_p[0] + 1;


  while(b != NULL){
    std::vector<std::string> records = b -> data;

    int stop = b -> records_count - 2;

    for (int i = 0; i < stop; i++){
      rec = records[i];

      zip_s = rec.substr(start_zip, length_zip);

      if(zip_s == zip){
        //std::cout « rec « "\n";

        state_s = rec.substr(start_state, length_state);
        place_s = rec.substr(start_place, length_place);

        std::cout « state_s « " " « place_s « "\n";
      }

    }
    b = b -> next;
  }

  delete(b);
}


void SequenceSet::nsew_most(std::string state){
  Block *b = first;
  std::string rec, rec_state, lat_s, long_s;
  float lat_f, long_f;

  std::vector<int> loc_zip = get_field_range_tuple(0);
  int start_zip = std::to_string(block_size).size() + loc_zip[0] - 1, length_zip = loc_zip[1] -
 loc_zip[0] + 1;

  std::vector<int> loc = get_field_range_tuple(2);
  int start_state = std::to_string(block_size).size() + loc[0] - 1, length_state = loc[1] - loc[0] +
 1;

  std::vector<int> loc_lat = get_field_range_tuple(4);
  int start_lat = std::to_string(block_size).size() + loc_lat[0] - 1, length_lat = loc_lat[1] -
 loc_lat[0] + 1;

  std::vector<int> loc_long = get_field_range_tuple(5);
  int start_long = std::to_string(block_size).size() + loc_long[0] - 1, length_long = loc_long[1] -
 loc_long[0] + 1;

  float east_most = 181;//= atof(get_field_from_record(5,0,0).c_str());
  float west_most = -181;//= atof(get_field_from_record(5,0,0).c_str());
  float north_most = -91;//= atof(get_field_from_record(4,0,0).c_str());
  float south_most = 91;//= atof(get_field_from_record(4,0,0).c_str());

  std::string zip_east_most;
  std::string zip_west_most;
  std::string zip_north_most;
  std::string zip_south_most;

  while(b != NULL){
    std::vector<std::string> records = b -> data;

    int stop = b -> records_count - 2;

    for (int i = 0; i < stop; i++){
      rec = records[i];
      rec_state = rec.substr(start_state, length_state);

      if(rec_state == state){
        //std::cout « rec « "\n";

        lat_s = rec.substr(start_lat,length_lat);
        long_s = rec.substr(start_long,length_long);

        lat_f = atof(lat_s.c_str());
        long_f = atof(long_s.c_str());


        if(lat_f < south_most){
          south_most = lat_f;
          zip_south_most = rec.substr();
        }
        if(lat_f > north_most){
          north_most = lat_f;
          zip_north_most = lat_f;
        }
        if(long_f <= east_most){
          east_most = long_f;
          zip_east_most = long_f;
        }
        if(long_f > west_most){
          west_most = long_f;
          zip_west_most = long_f;
        }

      }

    }


    b = b -> next;
  }

  std::cout « "\n\nNorth-most lat:" « north_most « "\n";
  std::cout « "South-most lat:" « south_most « "\n\n";
  std::cout « "East-most long:" « east_most « "\n";
  std::cout « "West-most long:" « west_most « "\n";
  delete(b);
```

```
}



/*


  bool found = false;

  while(b != NULL && !found){
    std::vector<std::string> records = b -> data;

    int stop = b -> records_count - 2;

    for (int i = 0; i < stop; i++){
      rec = records[i];
      rec_state = rec.substr(start_state, length_state);

      if(rec_state == state){
        lat_s = rec.substr(start_lat,length_lat);
        long_s = rec.substr(start_long,length_long);
        east_most= atof(long_s.c_str());
        west_most= atof(long_s.c_str());
        north_most= atof(lat_s.c_str());
        south_most= atof(lat_s.c_str());
      }

      if(found){
        i = stop;
      }

    }

    b = b -> next;
  }

  b = first;





void SequenceSet::nsew_most(std::string state){
  float east_most = 0.0;
  float west_most = 0.0;
  float north_most = 0.0;
  float south_most = 0.0;

  //std::string zipcode_east_most = 0;
  //std::string zipcode_west_most = 0;
  //std::string zipcode_north_most = 0;
  //std::string zipcode_south_most = 0;

  Block *copy = first;

  std::vector<int> loc = get_field_range_tuple(2);
  int start = std::to_string(block_size).size() + loc[0] - 1, length_state = loc[1] - loc[0] + 1;

  std::vector<int> loc_zip = get_field_range_tuple(0);
  int start_zip = std::to_string(block_size).size() + loc_zip[0] - 1, length_zip = loc_zip[1] -
 loc_zip[0] + 1;

  std::vector<int> loc_lat = get_field_range_tuple(4);
  int start_lat = std::to_string(block_size).size() + loc_lat[0] - 1, length_lat = loc_lat[1] -
 loc_lat[0] + 1;

  std::vector<int> loc_long = get_field_range_tuple(5);
  int start_long = std::to_string(block_size).size() + loc_long[0] - 1, length_long = loc_long[1] -
 loc_long[0] + 1;

  std::string lat_s, long_s, r, s, zip;

  while(copy != NULL){
    int i = 0;
    float lat_f, long_f;
    while(i < copy -> records_count){
      r = copy -> data[i];
      s = r.substr(start,length_state);
      if (s == state){
        std::cout « r « "\n";
        //find max of n, w, e, s
        lat_s = r.substr(start_lat,length_lat);
        long_s = r.substr(start_long,length_long);
        zip = r.substr(start_zip, length_zip);
        lat_f = atof(lat_s.c_str());
        long_f = atof(long_s.c_str());
        s = r.substr(start,length_state);
        std::cout « lat_f « "  -  " « long_f « "\n";
        if(lat_f <= south_most){
          south_most = lat_f;
          zipcode_south_most = zip;
        }
        if(lat_f > north_most){
          north_most = lat_f;
          zipcode_north_most = zip;
        }
        if(long_f <= east_most){
          east_most = long_f;
          zipcode_east_most = zip;
        }
        if(long_f > west_most){
          west_most = long_f;
          zipcode_west_most = r.substr(start_zip, length_zip);
        }
      }
      i++;
    }
    copy = copy -> next;
  }

  std::cout « "\n\nNorth-most lat:" « north_most « "\n";
  std::cout « "South-most lat:" « south_most « "\n\n";
  std::cout « "East-most long:" « east_most « "\n";
  std::cout « "West-most long:" « west_most « "\n";

  std::cout « "\n\nNorth-most zip-code:" « zipcode_north_most « "\n";
  std::cout « "South-most zip-code:" « zipcode_south_most « "\n\n";
  std::cout « "East-most zip-code:" « zipcode_east_most « "\n";
  std::cout « "West-most zip-code:" « zipcode_west_most « "\n";

}
```

4.8 SequenceSet.h File Reference

```
#include <iostream>
#include <string>
#include <iterator>
#include <vector>
```

Include dependency graph for SequenceSet.h: This graph shows which files directly or indirectly include this file:

Classes

- class SequenceSet
- struct SequenceSet::Block
- struct SequenceSet::Index

4.9 SequenceSet.h

```
/*
  Authors: Jacob Hopkins, Misky Abshir, Tyler Willard
  Date: 4/27/2020
*/
#include <iostream>
#include <string>
#include <iterator>
#include <vector>

/*
  This is a datatype for handling large file in and out of RAM.

  We need
```

```
    - constructor(s) / destructor (etc. for in-RAM objects)
    - create
    - open/load (necessary components of an existing SS {i.e. header record & index file into memory)
        consider optionally running the validate method
    - close
    - is_empty (via a flag, can be applied to either file, a block, a record slot in a block, or a field within a record)
    - search (for a record)
    - populate (populate the blocked record file from the input data file)
        consider populating to 3/4 capacity as a default parameter
        (can be changed for testing block merging, splitting, & record redistribution)
    - insert (a record)
    - delete (a record)
    - update (a field of a record)
    - display_record
    - diplay_SS (parameterized to display the whole record or a subset of fields)
    - validate (is your sequence set ordered by primary key? Can you get to each record via the index file?)
    - (...private helper functions/methods)
    - (...debug functions/methods) {consider using a static debug flag for the class}
*/
class SequenceSet
{
  private:
    struct Block;             //see below
    struct Index;
    Block *first;
    Index *root;
    int field_count;          //count of fields per record
    int block_size;           //records per block
    float default_cap;          //where the program will fill blocks to by default
    int record_size;          //number of characters per record
    int primary_key_index;
    std::string end_of_header;
    std::fstream in_file;
    std::ofstream out_file;
    std::string in_filename;          //filename for input
    std::string out_filename;         //filename for output
    std::vector<std::string> field_labels;  //labels of each field
    std::vector<std::string> field_sizes;          //sizes of each field
    std::vector<std::string> field_types;  //type for each field


  public:
    SequenceSet();
    SequenceSet(int b_size, int r_size, float d_cap, std::string i_filename, std::string o_filename);
    ~SequenceSet();
    void create();
    void load();
    void close();
    bool is_empty(int flag, int block, int record, int field);
    std::vector<int> search(std::string search_term);
    std::string get_field_from_record(int field, int record, int block);
    void populate();
    void insert(std::string new_record);
    void delete_record(int block, int record);
    void update(int block, int record, int field, std::string new_field);
    void display_record(int record, int block);
    void display_field(int field, int record, int block);
    void display_file(int limit);
    void display_SS();
    void validate();
    //void addIndex(int primKey, Block *b);                       /*! function prototype addIndex(int, Block) that adds an index in a record */
    //void delIndex(int primKey);                                 /*! function prototype delIndex(int) that removes an index in a record */
    void developer_show();
    int search_file(int primKey);
    std::vector<int> get_field_range_tuple(int field_index);
    void nsew_most(std::string state);
    void state_and_place_from_zip(std::string zip);
};



/*
  Here we create a Block

  block size {default to (512B / block)}

Each active block should include the following components:
    count of records ( > 0 )
    pointers to preceding & succeeding active blocks
    set of records ordered by key

Each avail block should include the following components:
    count of records ( == 0 )
    pointer to succeeding avail block

*/
struct SequenceSet::Block {
  Block *next, *previous;
  int records_count;

  std::vector<  std::string > data; //1 dimensional vector holding all records as 1 string
};

/*
  This is an index
*/
struct SequenceSet::Index {
  int key[4];
  Block *block[4];
  Index *subTree[4], *nextNode, *parent;
};
```

4.10 tester.cpp File Reference

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

Include dependency graph for tester.cpp:

Functions

- void menu ()
- int main ()

4.10.1 Function Documentation

4.10.1.1 menu() `void menu ( )`

Definition at line 6 of file tester.cpp.

4.10.1.2 main() `int main ( )`

Definition at line 23 of file tester.cpp.

Here is the call graph for this function:

4.11 tester.cpp

```
#include <iostream>
#include <string>
#include "SequenceSet.cpp"
using namespace std;

void menu(){
    cout«"----------------Sequence Set Generator---------------"«endl«endl;
    cout«"-----------------------------------------------------"«endl;
    cout « "We use lowercase letters to choose an optain" « endl;
    cout « "d: calls the delete method." « endl;
    cout « "i: calls the insert method." « endl;
    cout « "s: displays all the blocks." « endl;
    cout « "r: displays the record." « endl;
    cout « "f: displays the fields." « endl;
    cout « "b: displays the B+ Tree." « endl;
    cout « "c: creates the index file." « endl;
    cout « "u: calls the update method." « endl;
    cout « "m: will display this menu." « endl;
    cout « "x: will end this program." « endl;

}

int main()
{
    SequenceSet s;
    char choice =' ';

    s.populate();
    menu();
    while(choice!='x'){
        cout«"-----------------------------------------------------"«endl;
        cout«"Enter Choice"«endl;
        cin » choice;
        switch(choice){

            case 'd':   s.delete_record(),
                        s.create();
                        break;
            case 'm':   menu();

                        break;
            case 'r':   s.display_record();
                        break;
            case 'f':   s.display_field();
                        break;
            case 's':   s.display_SS();
                        break;
            case 'i':   s.insert(),
                        s.create();
                        break;
            case 'b':   b.print();
                        break;
            case 'c':   s.create();
                        break;
            case 'u':   s.update();
                        break;
            case 'x':   cout « "Terminating program, goodbye!" « endl;
        }
    }
    cout«"-----------------------------------------------------"«endl«endl;

    return 0;

}
```

4.12 testSequenceSet.cpp File Reference

```
#include <iostream>
#include <cstdio>
#include "SequenceSet.h"
```

Include dependency graph for testSequenceSet.cpp:

Functions

- int main (int arg_count, char ∗∗arg_values)

4.12.1 Function Documentation

4.12.1.1 main()

```
int main (
    int arg_count,
    char ** arg_values )
```

Definition at line 24 of file testSequenceSet.cpp.

4.13 testSequenceSet.cpp

```
00001 /*
00002     Authors: Jacob Hopkins, Misky Abshir, and Tyler Willard
00003     Date: 4/27/2020
00004 
00005     testSequenceSet.cpp
00006     This is a test program.
00007 
00008     Sequence set is a class to handle reading data into and out of files and processing with
      performance.
00009 
00010     This program is to show the functionality of the SequenceSet class found in 'SequenceSet.h'
00011 
00012     In video 4:
00013     14:20
00014 
00015 */
00016 
00017 #include<iostream>
00018 #include <cstdio>
00019 #include "SequenceSet.h"
00020 
00021 /*
00022     Here is the main function of the test driver.
00023 */
00024 int main(int arg_count, char** arg_values){
00025 
00026     //show of arguments and example using them
00027     std::cout « "You have entered " « arg_count « " arguments:" « "\n";
00028     for (int i = 0; i < arg_count; ++i)
00029         std::cout « arg_values[i] « "\n";
00030 
00031 
00032     //introduction
00033     std::cout « "This is the test program for the SequenceSet class. " « std::endl;
00034 
00035 
00036     /*
00037         Here we declare a SequenceSet named test.
00038     */
00039     std::cout « "Declaring SequenceSet: test" « std::endl;
00040     //SequenceSet test;
00041     std::cout « "Declaring SequenceSet complete." « std::endl « std::endl;
00042 
00043 
00044     /*
00045         Here we initialize the sequence set test.
00046     */
00047     std::cout « "Initalizing test" « std::endl;
00048     //test = SequenceSet();
00049     std::cout « "test initalization complete." « std::endl « std::endl;
00050 
00051     /*
00052 
00053     */
00054     std::cout « "" « std::endl;
00055 
00056     std::cout « " complete." « std::endl « std::endl;
00057 
00058 
00059 
00060     //forbiden code here
00061     //wait for character so the screen does not disappear
00062     getchar();
00063 
00064     //return that the program ran correctly
00065     return 0;
00066 }
00067 
```

4.14 us_postal_codes_column_reorder.txt File Reference

4.15 us_postal_codes_formatted.txt File Reference

4.16 us_postal_codes_row_randomized.txt File Reference

4.17 us_postal_codes_sequence_set_file.txt File Reference

Index