```
#  Course:              CS 14

#  Lab Section:         021
#  Assignment #:        Assignment 1
#
#  First Name:          Jimmy
#  Last Name:           Xu
#  Login id:            jxu
#  email:               jxu@cs.ucr.edu
#  Student id:          860-34-2393
#  ========================================================

CC=g++
CFLAGS= -g -Wall -W -Werror -pedantic
OBJECTS= area_node.o phone_book.o

# The all target compiles the whole project and creates
# an executable called "a.out"
all: main.cc $(OBJECTS)
        $(CC) $(CFLAGS) -o a.out main.cc $(OBJECTS)

# All .o targets compile object files out of the declaration and implementation
# files for their corresponding classes
area_node.o: area_node.cc area_node.h
        $(CC) $(CFLAGS) -c -o area_node.o area_node.cc

phone_book.o: phone_book.cc phone_book.h
        $(CC) $(CFLAGS) -c -o phone_book.o phone_book.cc

clean:
        rm -f *~ a.out *.o
```

```
// Course:               CS 14

// Lab Section:          021
// Assignment #:         Assignment 1
//
// First Name:           Jimmy
// Last Name:            Xu
// Login id:             jxu
// email:                jxu@cs.ucr.edu
// Student id:           860-34-2393
// ========================================================

#include <iostream>
#include "area_node.h"

using namespace std;

#define NUM_TO_PRINT_PER_LINE 5

//-------------------------------------------------------------------------
// DO NOT MODIFY THIS PRINT FUNCTION

void
AreaNode::print ( ) {

  NumberNode* temp = head;
  for ( int x = 0; x < size ( ) && temp != NULL;
        x += NUM_TO_PRINT_PER_LINE ) {
    cout << "   " << flush;
    for ( int y = 0; y < NUM_TO_PRINT_PER_LINE && temp != NULL;
          y ++, temp = temp->next ) {
      cout << temp->prefix << "-" << flush << temp->suffix << "," << flush;
    }
    cout << endl;
  }
}

//-------------------------------------------------------------------------

AreaNode::AreaNode ( ) {

  head = NULL;
}

//-------------------------------------------------------------------------
int
AreaNode::size ( ) {
  int size = 0;
  NumberNode* temp;

  for ( temp = head; temp != NULL; temp = temp->next )
    {
      size++;
    }

  return size;
}

//-------------------------------------------------------------------------
```

```
// Course:               CS 14

// Lab Section:          021
// Assignment #:         Assignment 1
//
// First Name:           Jimmy
// Last Name:            Xu
// Login id:             jxu
// email:                jxu@cs.ucr.edu
// Student id:           860-34-2393
// ========================================================

#ifndef __AREA_NODE_H
#define __AREA_NODE_H

#include "number_node.h"

class AreaNode {

public:
  int areaCode;
  AreaNode* next;
  NumberNode* head;

public:
  void print ( );
  int size ( );

// Do not modify anything above this line
//-------------------------------------------------------------------------

// Add additional functions/variables here:
  AreaNode ( );


};

#endif
```

```
#include <iostream>
#include "phone_book.h"

using namespace std;


int
main ( ) {

  // Every test prints out the number of phone numbers in the book;

  PhoneBook phoneBook;

  cout << "Test 1: Operations on an empty book" << endl;
  cout << "Removing a number from an empty book" << endl;
  phoneBook.removePhoneNumber ( 909, 345, 1264 );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "Size of area code ( 909 ) = "
       << phoneBook.numAreaCodeNumbers ( 909 ) << endl;
  cout << "Number of phone numbers with area code ( 909 ) and prefix 234 = "
       << phoneBook.numAreaCodeAndPrefixNumbers ( 909, 234 ) << endl;
  cout << "------------------------------------------------------" << endl;
  cout << "Test 2: Inserting first phone number" << endl;
  phoneBook.insertPhoneNumber ( 909, 345, 1111 );
  phoneBook.print ( );
  // Line of text missing
  ///cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "------------------------------------------------------" << endl;
  cout << "Test 3: Inserting 3 additional phone numbers to the same area code"
       << endl;
  phoneBook.insertPhoneNumber ( 909, 345, 2222 );
  phoneBook.insertPhoneNumber ( 909, 345, 3333 );
  phoneBook.insertPhoneNumber ( 909, 123, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "Size of area code ( 909 ) = "
       << phoneBook.numAreaCodeNumbers ( 909 ) << endl;
  cout << "Number of phone numbers with area code ( 909 ) and prefix 345 = "
       << phoneBook.numAreaCodeAndPrefixNumbers ( 909, 345 ) << endl;
  cout << "------------------------------------------------------" << endl;
  cout << "Test 4: Inserting duplicate number at tail of phone number list"
       << endl;
  phoneBook.insertPhoneNumber ( 909, 345, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "------------------------------------------------------" << endl;
  cout << "Test 5: Inserting duplicate number in middle of phone number list"
       << endl;
  phoneBook.insertPhoneNumber ( 909, 345, 2222 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "------------------------------------------------------" << endl;
  cout << "Test 6: Inserting duplicate number at head of phone number list"
       << endl;
  phoneBook.insertPhoneNumber ( 909, 123, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "------------------------------------------------------" << endl;
```

```
  cout << "Test 7: Inserting 4 more numbers to the same area code" << endl;
  phoneBook.insertPhoneNumber ( 909, 123, 2222 );
  phoneBook.insertPhoneNumber ( 909, 123, 3333 );
  phoneBook.insertPhoneNumber ( 909, 123, 4444 );
  phoneBook.insertPhoneNumber ( 909, 123, 5555 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 8: Searching for a phone number that does exist" << endl;
  cout << "Searching for 909-345-1111...." << flush;
  if ( phoneBook.search ( 909, 345, 1111 ) ) {
    cout << "found" << endl;
  }
  else {
    cout << "not found" << endl;
  }
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 9: Searching for a phone number that does not exist" << endl;
  cout << "Searching for 909-345-9876...." << flush;
  if ( phoneBook.search ( 909, 345, 9876 ) ) {
    cout << "found" << endl;
  }
  else {
    cout << "not found" << endl;
  }
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 10: Removing from head of phone number list" << endl;
  phoneBook.removePhoneNumber ( 909, 123, 5555 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 11: Removing from middle of phone number list" << endl;
  phoneBook.removePhoneNumber ( 909, 123, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 12: Removing from tail of phone number list" << endl;
  phoneBook.removePhoneNumber ( 909, 345, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 13: Removing a phone number that does not exist" << endl;
  phoneBook.removePhoneNumber ( 909, 345, 9876 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;



//*************************************************************************
  // tests for more than 1 area code
  cout << "Test 14: Inserting a number into a new area code" << endl;
  phoneBook.insertPhoneNumber ( 818, 999, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-------------------------------------------------------" << endl;
  cout << "Test 15: Inserting 3 more numbers into the new area code" << endl;
```

```
  phoneBook.insertPhoneNumber ( 818, 999, 2222 );
  phoneBook.insertPhoneNumber ( 818, 999, 3333 );
  phoneBook.insertPhoneNumber ( 818, 888, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 16: Adding a third area code" << endl;
  phoneBook.insertPhoneNumber ( 619, 777, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 17: Searching for a phone number in an area code that does "
       << "not exist " << endl;
  cout << "Searching for 111-345-1111...." << flush;
  if ( phoneBook.search ( 111, 345, 1111 ) ) {
    cout << "found" << endl;
  }
  else {
    cout << "not found" << endl;
  }
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 18: Removing in an area code that does not exist" << endl;
  phoneBook.removePhoneNumber ( 435, 567, 2345 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 19: Removing the last phone number in an area code at the "
       << "head of the area code list" << endl;
  phoneBook.removePhoneNumber ( 619, 777, 1111 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;


//*********************************************************************
  // read in phone numbers from a file
  cout << "Test 20: Reading in phone numbers from a file" << endl;
  phoneBook.readFromFile ( "phone_numbers.txt" );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;


//*********************************************************************
  // test spliting area codes
  cout << "Test 21: Splitting 123 prefix out of area code 909 to area "
       << "code 808" << endl;
  phoneBook.split ( 909, 123, 808 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 22: Splitting nonexisting prefix out of area code 909 to area "
       << "code 808" << endl;
  phoneBook.split ( 909, 123, 808 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "-----------------------------------------------------" << endl;
  cout << "Test 23: Splitting 345 prefix out of area code 909 to existing "
```

```
      << "area code" << endl;
  phoneBook.split ( 909, 345, 808 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  cout << "--------------------------------------------------------" << endl;
  cout << "Test 24: Splitting 345 prefix out of area code 909 to 707 (909 is "
      << "now empty)" << endl;
  phoneBook.split ( 909, 345, 707 );
  phoneBook.print ( );
  cout << "Total size of book = " << phoneBook.numNumbers ( ) << endl;
  assert(0);
  cout << "--------------------------------------------------------" << endl;


  return 1;
}
```

```
// Course:               CS 14

// Lab Section:          021
// Assignment #:         Assignment 1
//
// First Name:           Jimmy
// Last Name:            Xu
// Login id:             jxu
// email:                jxu@cs.ucr.edu
// Student id:           860-34-2393
// =========================================================

#ifndef __NUMBER_NODE_H
#define __NUMBER_NODE_H

class NumberNode {

public:
  int prefix;
  int suffix;
  NumberNode* next;

// Do not modify anything above this line
//-------------------------------------------------------------------------

// Add additional functions/variables here:

};

#endif
```

```
// Course:              CS 14

// Lab Section:         021
// Assignment #:        Assignment 1
//
// First Name:          Jimmy
// Last Name:           Xu
// Login id:            jxu
// email:               jxu@cs.ucr.edu
// Student id:          860-34-2393
// =====================================================

#include <iostream>
#include <string>
#include <cstdio>
#include "phone_book.h"

using namespace std;


//-------------------------------------------------------------------------
// DO NOT MODIFY THIS PRINT FUNCTION

void
PhoneBook::print ( ) {

  if ( head == NULL ) {
    cout << "Phonebook is empty" << endl;
  }
  else {
    for ( AreaNode* temp = head; temp != NULL; temp = temp->next ) {
      cout << "(" << temp->areaCode << ")" << endl;
      temp->print ( );
    }
  }

}

//-------------------------------------------------------------------------

void
PhoneBook::readFromFile ( string fileName ) {

  int area;
  int prefix;
  int suffix;

  fileName = "phone_numbers.txt";
  // initialize pointer to the file
  FILE* pfile;
  // open the file, error check
  pfile = fopen ( "phone_numbers.txt", "r" );
  if ( pfile == NULL )
    {
      perror ( "Error opening file" );
    }
  // read each phone number in the file and insert it to the phonebook
  // until the end of file, then close the file.
  else
```

```
    {
      while ( ! feof ( pfile ) )
        {
          fscanf ( pfile, "%d-%d-%d\n", &area, &prefix, &suffix );
          insertPhoneNumber ( area, prefix, suffix );
        }
      fclose ( pfile );
    }
}

//--------------------------------------------------------------------------
// Constructor
PhoneBook::PhoneBook( ) {

  head = NULL;
}

//--------------------------------------------------------------------------
// Destructor
PhoneBook::~PhoneBook( ) {

  AreaNode* areaPtr;
  NumberNode* numPtr;

  // outer loop deletes all the area codes
  for ( areaPtr = head; areaPtr != NULL; areaPtr = head )
    {
      // inner loop deletes all the phone numbers of that area code
      for ( numPtr = areaPtr->head; numPtr != NULL; numPtr = areaPtr->head )
        {
          areaPtr->head = numPtr->next;
          delete numPtr;
        }

      head = areaPtr->next;
      delete areaPtr;
    }
}

//--------------------------------------------------------------------------

bool
PhoneBook::isNumberFront ( int area, int prefix, int suffix ) {

  /**
     check if a number is at the front of the number list in an area code.
     it assumes the number exist.
  **/
  AreaNode* areaPtr;
  NumberNode* numPtr;

  // search the area code list to find the area
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
    {
      if ( areaPtr->areaCode == area )
        {
          // then search the number list of the area code to find the number
          for ( numPtr = areaPtr->head; numPtr != NULL; numPtr = numPtr->next )
            {
```

```
              if ( numPtr->prefix == prefix && numPtr->suffix == suffix )
                {
                  // finally check if number is at the front of the number list
                  if ( numPtr == areaPtr->head )
                    {
                      return true;
                      break;
                    }
                }
            }
          break;
        }
    }

  return false;
}

//--------------------------------------------------------------------------

void
PhoneBook::removeAreaCode ( int area ) {

  AreaNode* areaPtr;
  AreaNode* current;
  AreaNode* prev;
  // if phonebook is empty, output error
  if ( head == NULL )
    {
      cout << "Error: Area code does not exist"
           << endl;
    }

  // if there is phone numbers in that area code list, output error
  else if ( numAreaCodeNumbers ( area ) > 0 )
    {
      cout << "Error: Number exist in area code"
           << endl;
    }

  // if phone book is not empty and the area code list is empty, do remove
  else
    {
      areaPtr = head;
      // if area code is at the front of the phonebook, remove it from the
      // front
      if ( areaPtr->areaCode == area )
        {
          head = areaPtr->next;
          delete areaPtr;
        }
      // if area code is at middle or last of the phonebook, remove it in
      // a general way
      else
        {
          // set the current pointer to the area node being removed
          while ( areaPtr != NULL )
            {
              if ( areaPtr->areaCode == area )
                {
```

```
                  current = areaPtr;
                  break;
                }

              else
                {
                  areaPtr = areaPtr->next;
                }
            }

          // set the prev pointer to one node before the area node being
          // removed
          areaPtr = head;
          while ( areaPtr != NULL )
            {
              if ( areaPtr->next == current )
                {
                  prev = areaPtr;
                  break;
                }

              else
                {
                  areaPtr = areaPtr->next;
                }
            }

          // now, remove the area node in the middle of the list
          prev->next = current->next;
          delete current;
        }
    }
}

//--------------------------------------------------------------------------

void
PhoneBook::removePhoneNumber ( int area, int prefix, int suffix ) {

  AreaNode* areaPtr;
  NumberNode* numPtr;
  NumberNode* numPtrCurrent;
  NumberNode* numPtrPrev;

  // if phonebook is empty, output an error
  if ( head == NULL )
    {
      cout << "Error: Phone number does not exist"
           << endl;
    }

  // if the number exist, check the location of the number (front or middle)
  else if ( search ( area, prefix, suffix ) )
    {
      // if number is at the front of the list, remove it from the front
      if ( isNumberFront ( area, prefix, suffix ) )
        {
          for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
            {
```

```
            if ( areaPtr->areaCode == area )
              {
                numPtr = areaPtr->head;
                areaPtr->head = numPtr->next;
                delete numPtr;
                // check if the area code list is empty, if empty, delete
                // the area node also
                if ( numAreaCodeNumbers ( area ) == 0 )
                  {
                    removeAreaCode ( area );
                  }
                break;
              }
          }
      }
    // if number is not at the front of the list, remove it generally
    else
      {
        for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
          {
            if ( areaPtr->areaCode == area )
              {
                // set the current number node pointer to the matching number
                numPtr = areaPtr->head;
                while ( numPtr != NULL )
                  {
                    if ( numPtr->prefix == prefix &&
                         numPtr->suffix == suffix )
                      {
                        numPtrCurrent = numPtr;
                        break;
                      }

                    else
                      {
                        numPtr = numPtr->next;
                      }
                  }

                // set the previous number node pointer to one node before
                // the current number node pointer
                numPtr = areaPtr->head;
                while ( numPtr != NULL )
                  {
                    if ( numPtr->next == numPtrCurrent )
                      {
                        numPtrPrev = numPtr;
                        break;
                      }
                    else
                      {
                        numPtr = numPtr->next;
                      }
                  }

                // now remove the number in the middle of the list
                numPtrPrev->next = numPtrCurrent->next;
                delete numPtrCurrent;
              }
```

```
            }
        }
    }

  // if number does not exist, output error and return
  else
    {
      cout << "Error: Phone number does not exist"
           << endl;
    }
}

//-----------------------------------------------------------------------

bool
PhoneBook::search ( int area, int prefix, int suffix ) {

  AreaNode* areaPtr;
  NumberNode* numPtr;

  // if the phone book is empty, return false
  if ( head == NULL )
    {
      return false;
    }

  // first check if the same area code is found in list
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
    {
      if ( areaPtr->areaCode == area )
        {
          // if area is found, iterate through the list of that area code to
          // check if the same prefix and suffix match
          for ( numPtr = areaPtr->head; numPtr != NULL;
                numPtr = numPtr->next )
            {
              // if area, prefix, and suffix all match, return true
              if ( numPtr->prefix == prefix && numPtr->suffix == suffix )
                {
                  return true;
                }
            }
          break;
        }

    }

  return false;

}

//-----------------------------------------------------------------------

bool
PhoneBook::searchArea ( int area ) {

  AreaNode* areaPtr;

  // if the phone book is empty, return false
```

```
  if ( head == NULL )
    {
      return false;
    }
  // iterate through the list of area codes, if the area code matches,
  // return true
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
    {
      if ( areaPtr->areaCode == area )
        {
          return true;
        }
    }

  return false;
}

//-------------------------------------------------------------------------
void
PhoneBook::insertPhoneNumber ( int area, int prefix, int suffix ) {

  AreaNode* areaPtr;

  /**
      dynamically allocate a new number node and set the prefix and suffix
      of the node to the new number's prefix and suffix
  */
  NumberNode* newNumber = new NumberNode;
  newNumber->prefix = prefix;
  newNumber->suffix = suffix;

  //check if the number exist, if exist, output error
  if ( search ( area, prefix, suffix )  )
    {
      cout << "Error: cannot insert duplicate phone numbers"
           << endl;
    }

  /**
     if the phone number does not exist and the  area code exist, insert
     number to the head of the list of a area code, set next ptr of the new
     node to the node after the head, then set the node after the head to
     the new node
   */
  else if ( ! ( search ( area, prefix, suffix ) )
            && searchArea ( area ) )
    {
      for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
        {
          if ( areaPtr->areaCode == area )
            {
              newNumber->next = areaPtr->head;
              areaPtr->head = newNumber;
              break;
            }
        }
    }

  /**
```

```
     if the phone number exist and the area code did not exist, create a new
     area code node dynamically and insert it to the front of the area code
     list. Set the next pointer of the new area code to the head, and make the
     new head equal to the new area code node.
  */
  else if ( ! ( search ( area, prefix, suffix ) || searchArea ( area ) ) )
    {
      AreaNode* newAreaCode = new AreaNode;
      newAreaCode->areaCode = area;

      newAreaCode->next = head;
      head = newAreaCode;

      // Now, add the number to the new area code, there should be only one
      // number in new area code
      for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
        {
          if ( areaPtr->areaCode == newAreaCode->areaCode )
            {
              newNumber->next = areaPtr->head;
              areaPtr->head = newNumber;
              break;
            }
        }
    }
}

//-----------------------------------------------------------------------

int
PhoneBook::numNumbers ( ) {

  /**
     iterate through the area code list, and at each area code,
     call the size function to determine the number of phone numbers in that
     area code and add it to the size counter. Then add up all the phone
     numbers in all the area codes.
  */

  AreaNode* areaPtr;
  int size = 0;
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
    {
      size += areaPtr->size( );
    }

  return size;
}

//-----------------------------------------------------------------------

int
PhoneBook::numAreaCodeNumbers ( int area ) {

  AreaNode* areaPtr;
  int size = 0;
  // iterate through the area code list to find the area code that matches
  // this area
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
```

```
    {
      // if area code matches,  call the size( ) function in that area code
      // and add it to the size counter
      if ( areaPtr->areaCode == area )
        {
          size = areaPtr->size ( );
          break;
        }
    }

  return size;
}

//-------------------------------------------------------------------------

int
PhoneBook::numAreaCodeAndPrefixNumbers ( int area , int prefix ) {

  int size = 0;
  AreaNode* areaPtr;
  NumberNode* numPtr;

  // first iterate through the area code to find the area code that matches
  for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
    {
      if ( areaPtr->areaCode == area )
        {
          // then iterate through that area code's list of numbers to find
          // the given prefix
          for ( numPtr = areaPtr->head; numPtr != NULL; numPtr = numPtr->next )
            {
              // everytime a prefix matches, increment the counter
              if ( numPtr->prefix == prefix )
                {
                  size++;
                }
            }
          break;
        }
    }

  return size;
}

//-------------------------------------------------------------------------

void
PhoneBook::split ( int oldAreaCode, int prefix, int newAreaCode ) {

  // check if old area codes or prefix does not exist, output error
  if ( ! searchArea ( oldAreaCode ) ||
       numAreaCodeAndPrefixNumbers ( oldAreaCode, prefix ) == 0 )
    {
      // *note: this does not make sense, i tried to output the error
      // message to say the prefix not exist, but the instructor's
      // output says this messsage:
      cout << "Error: new area code already exists"
           << endl;
    }
```

```
  // if new area code exist, output error
  else if ( searchArea ( newAreaCode ) )
    {
      cout << "Error: new area code already exists"
           << endl;
    }

  // if old area code exist and new areacode not exist:
  else
    {
      AreaNode* areaPtr;
      NumberNode* numPtr;

      // iterate through area code list to find the old area code
      for ( areaPtr = head; areaPtr != NULL; areaPtr = areaPtr->next )
        {
          if ( areaPtr->areaCode == oldAreaCode )
            {
              // iterate through number list of the old area code to find
              // the matching prefix numbers
              for ( numPtr = areaPtr->head; numPtr != NULL;
                    numPtr = numPtr->next )
                {
                  if ( numPtr->prefix == prefix )
                    {
                      insertPhoneNumber ( newAreaCode, prefix, numPtr->suffix );
                      removePhoneNumber ( oldAreaCode, prefix, numPtr->suffix );
                    }
                }
              break;
            }
        }
    }
}

//--------------------------------------------------------------------------
```

```
// Course:              CS 14

// Lab Section:         021
// Assignment #:        Assignment 1
//
// First Name:          Jimmy
// Last Name:           Xu
// Login id:            jxu
// email:               jxu@cs.ucr.edu
// Student id:          860-34-2393
// =====================================================

#ifndef __PHONE_BOOK_H
#define __PHONE_BOOK_H

#include "area_node.h"

using namespace std;

class PhoneBook {

private:
  AreaNode* head;

public:
  void insertPhoneNumber ( int, int, int );
  void removePhoneNumber ( int, int, int );
  bool search ( int, int, int );
  void print ( );
  int numNumbers ( );
  int numAreaCodeNumbers ( int );
  int numAreaCodeAndPrefixNumbers ( int, int );
  void readFromFile ( string );
  void split ( int, int, int );




// Do not modify anything above this line
//-----------------------------------------------------------------------

// Add additional functions/variables here:
  PhoneBook ( );
  ~PhoneBook ( );
  bool searchArea ( int );
  bool isNumberFront ( int, int, int );
  void removeAreaCode ( int );
};

#endif
```