Course: ENSF614 – Fall 2025
Lab #: Lab 1
Instructor: Mahmood Moussavi
Student Name: John Zhou
Submission Date: Sep 17th, 2025

I have been keeping all the files in github. I hope by providing this github link will help you a little bit. https://github.com/JZ-Zhou-UofC/ENSF-614-assignment-repo

Exercise B

```
/*
*
*
File Name: dictionaryList.cpp
Assignment: Lab 1 Exercise B
*  Completed by: John
*  Submission Date: Sept 17, 2025
*/
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"
using namespace std;

Node::Node(const int &keyA, const Datum &datumA, Node *nextA)
  : keyM(keyA), datumM(datumA), nextM(nextA)
{
}

DictionaryList::DictionaryList()
  : sizeM(0), headM(0), cursorM(0)
{
}
```

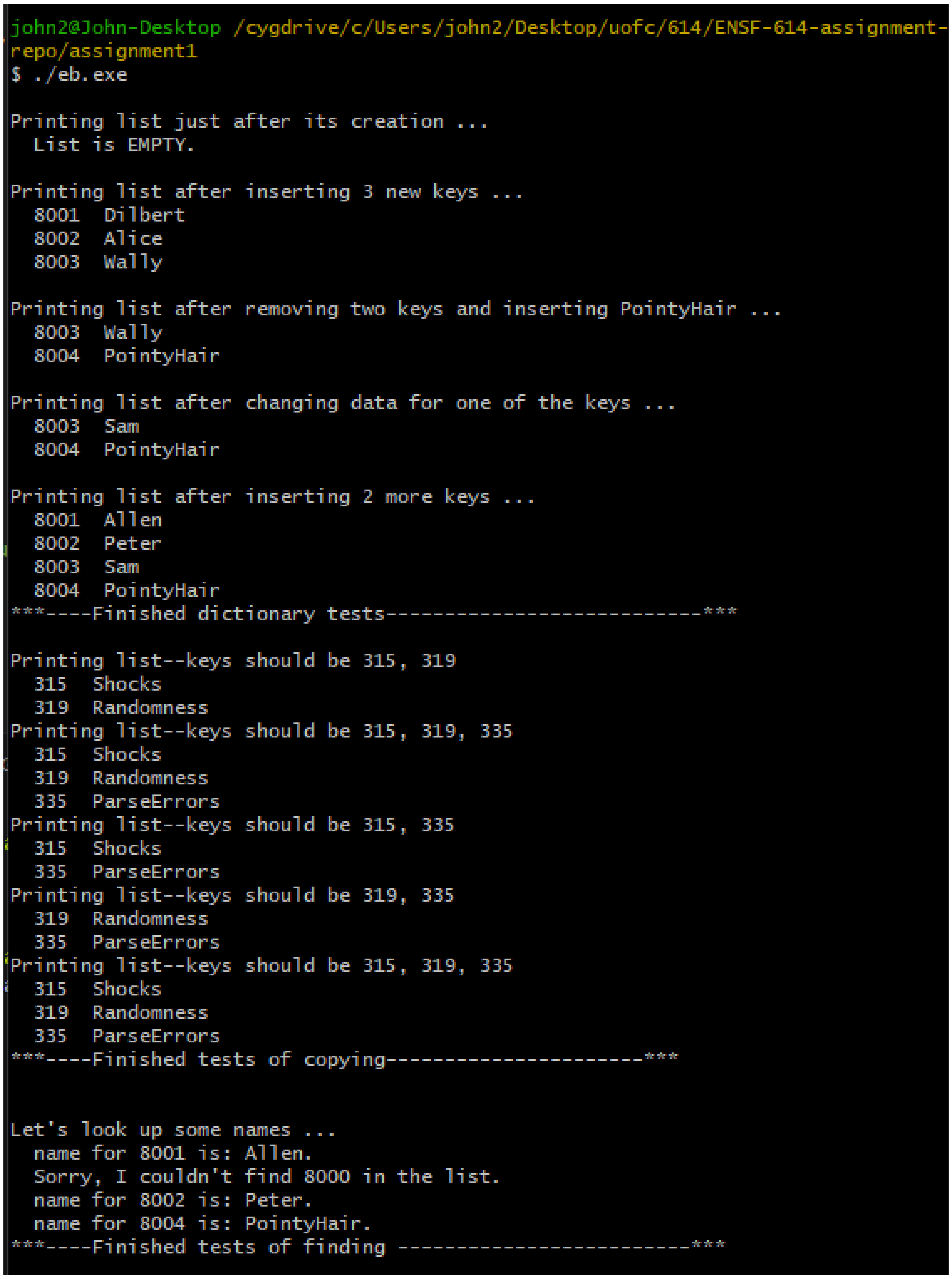
```
int DictionaryList::size() const
{
   return sizeM;
}

int DictionaryList::cursor_ok() const
{
   return cursorM != 0;
}

const int &DictionaryList::cursor_key() const
{
   assert(cursor_ok());
   return cursorM->keyM;
}

const Datum &DictionaryList::cursor_datum() const
{
   assert(cursor_ok());
   return cursorM->datumM;
}

void DictionaryList::insert(const int &keyA, const string &datumA)
{
   // Add new node at head?
   if (headM == 0 || keyA < headM->keyM)
   {
      headM = new Node(keyA, datumA, headM);
      sizeM++;
```

```
}

// Overwrite datum at head?
else if (keyA == headM->keyM)
   headM->datumM = datumA;

// Have to search ...
else
{

   // POINT ONE

   // if key is found in list, just overwrite data;
   for (Node *p = headM; p != 0; p = p->nextM)
   {
      if (keyA == p->keyM)
      {
         p->datumM = datumA;
         return;
      }
   }

   // OK, find place to insert new node ...
   Node *p = headM->nextM;
   Node *prev = headM;

   while (p != 0 && keyA > p->keyM)
   {
      prev = p;
```

```
            p = p->nextM;
        }

        prev->nextM = new Node(keyA, datumA, p);
        sizeM++;
    }
    cursorM = NULL;
}

void DictionaryList::remove(const int &keyA)
{
    if (headM == 0 || keyA < headM->keyM)
        return;

    Node *doomed_node = 0;

    if (keyA == headM->keyM)
    {
        doomed_node = headM;
        headM = headM->nextM;

        // POINT TWO
    }
    else
    {
        Node *before = headM;
        Node *maybe_doomed = headM->nextM;
        while (maybe_doomed != 0 && keyA > maybe_doomed->keyM)
        {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->nextM;
        }

        if (maybe_doomed != 0 && maybe_doomed->keyM == keyA)
        {
            doomed_node = maybe_doomed;
            before->nextM = maybe_doomed->nextM;
        }
    }
    if (doomed_node == cursorM)
        cursorM = 0;

    delete doomed_node; // Does nothing if doomed_node == 0.
    sizeM--;
}

void DictionaryList::go_to_first()
{
    cursorM = headM;
}

void DictionaryList::step_fwd()
{
    assert(cursor_ok());
    cursorM = cursorM->nextM;
}

// The following functions are supposed to be completed by the stuents, as part
// of the exercise B. the given code for this fucntion are just place-holders

// in order to allow successful linking when you're esting insert and remove.

// Replace them with the definitions that work.


DictionaryList::DictionaryList(const DictionaryList &source) : sizeM(0), headM(0), cursorM(0)

{


  if (source.headM == 0)

  {


    return;

  }

  headM = new Node(source.headM->keyM, source.headM->datumM, nullptr);

  sizeM = 1;

  if (source.cursorM == source.headM)

    cursorM = headM;

  Node *currentNode = headM;

  Node *sourceNextNode = source.headM->nextM;

  while (sourceNextNode)

  {

    currentNode->nextM = new Node(sourceNextNode->keyM, sourceNextNode->datumM, nullptr);

    currentNode = currentNode->nextM;

    sizeM++;

    if (source.cursorM != nullptr && source.cursorM == sourceNextNode)

    {

      cursorM = currentNode;

    }

    sourceNextNode = sourceNextNode->nextM;

  }
```

```
}

DictionaryList &DictionaryList::operator=(const DictionaryList &rhs)
{
    if (this == &rhs)
    {
        return *this;
    }
    make_empty();
    if (rhs.headM == 0)
    {
        headM = 0;
        cursorM = 0;
        sizeM = 0;
        return *this;
    }
    headM = new Node(rhs.headM->keyM, rhs.headM->datumM, nullptr);
    sizeM = 1;
    if (rhs.cursorM == rhs.headM)
        cursorM = headM;
    Node *currentNode = headM;
    Node *rhsNextNode = rhs.headM->nextM;
    while (rhsNextNode)
    {
        currentNode->nextM = new Node(rhsNextNode->keyM, rhsNextNode->datumM, nullptr);
        currentNode = currentNode->nextM;
        sizeM++;
        if (rhs.cursorM != nullptr && rhs.cursorM == rhsNextNode)
        {
            cursorM = currentNode;
        }
        rhsNextNode = rhsNextNode->nextM;
    }
    return *this;
}

DictionaryList::~DictionaryList()
{

    make_empty();
}

void DictionaryList::find(const int &keyA)
{
    Node *current = headM;

    while (current)
    {
        if (current->keyM == keyA)
        {
            cursorM = current;
            return;
        }
        if (current->keyM > keyA)
        {
            break;
        }
        current = current->nextM;
    }
    cursorM = nullptr;
}

void DictionaryList::make_empty()
{
    Node *current = headM;
    while (current)
    {
        Node *next = current->nextM;
        delete current;
        current = next;
    }
    headM = 0;
    cursorM = 0;
    sizeM = 0;
}
```

Screenshot:



```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/614/ENSF-614-assignment-
repo/assignment1
$ ./eb.exe

Printing list just after its creation ...
  List is EMPTY.

Printing list after inserting 3 new keys ...
  8001  Dilbert
  8002  Alice
  8003  Wally

Printing list after removing two keys and inserting PointyHair ...
  8003  Wally
  8004  PointyHair

Printing list after changing data for one of the keys ...
  8003  Sam
  8004  PointyHair

Printing list after inserting 2 more keys ...
  8001  Allen
  8002  Peter
  8003  Sam
  8004  PointyHair
***----Finished dictionary tests---------------------------***

Printing list--keys should be 315, 319
  315  Shocks
  319  Randomness
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 335
  315  Shocks
  335  ParseErrors
Printing list--keys should be 319, 335
  319  Randomness
  335  ParseErrors
Printing list--keys should be 315, 319, 335
  315  Shocks
  319  Randomness
  335  ParseErrors
***----Finished tests of copying----------------------***


Let's look up some names ...
  name for 8001 is: Allen.
  Sorry, I couldn't find 8000 in the list.
  name for 8002 is: Peter.
  name for 8004 is: PointyHair.
***----Finished tests of finding -------------------------***
```

Exercise D

```
/*
*
*
File Name: Human.cpp
Assignment: Lab 1 Exercise D
*  Completed by: John Zhou
*  Submission Date: Sept 17, 2025
*/
#include "Human.h"
#include <cstring>
#include <iostream>
using namespace std;

Human::Human(const char *nam, double x, double y)
    : location(x, y)
{
    name = new char[strlen(nam) + 1];
    strcpy(name, nam);
}

Human::Human(const Human &other)
    : location(other.location)
{
    name = new char[strlen(other.name) + 1];
    strcpy(name, other.name);
}

Human &Human::operator=(const Human &rhs)
{
   if (this != &rhs)
   {
      location = rhs.location;

      delete[] name;
      name = new char[strlen(rhs.name) + 1];
      strcpy(name, rhs.name);
   }
   return *this;
}

Human::~Human()
{
   delete[] name;
}

const char *Human::get_name() const { return name; }
Point Human::get_point() const { return location; }

void Human::set_name(const char *nam)
{
   delete[] name;
   name = new char[strlen(nam) + 1];
   strcpy(name, nam);
}
```

```
void Human::display() const
{
   cout << "Human Name: " << name
        << "\nHuman Location: "
        << location.get_x() << " ,"
        << location.get_y() << ".\n"
        << endl;
}
```

```
/*
*
*
File Name: Human.H
Assignment: Lab 1 Exercise D
*  Completed by: John Zhou
*  Submission Date: Sept 17, 2025
*/
#ifndef HUMAN_H
#define HUMAN_H

#include "Point.h"

class Human {
protected:
   Point location;   // Location of an object of Human on a Cartisian Plain
   char *name;       // Human's name

public:
   // Constructor

   Human(const char* nam = "", double x = 0, double y = 0);

   //the name will be created in the run time. Copy constructor, assignment operater and destructor are
needed
   Human(const Human& other);

   Human& operator=(const Human& rhs);
```

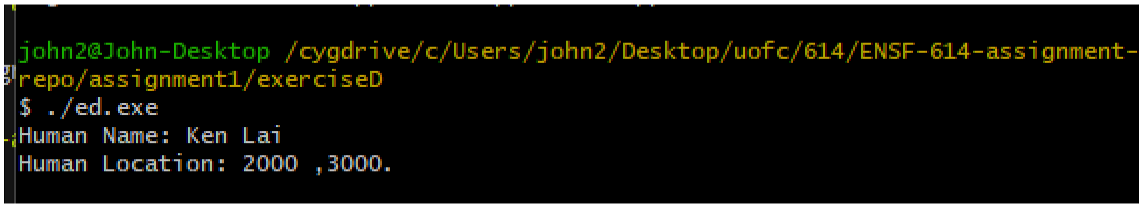
```
    ~Human();


    // this getter should not have the ability to change anything inside the class
    // this should return a const char pointer for safety.
    const char* get_name() const;
    Point get_point() const;



    void set_name(const char* nam);



    void display() const;
};

#endif
```

```
/*
*
*
File Name: main.cpp
Assignment: Lab 1 Exercise D
*  Completed by: John Zhou
*  Submission Date: Sept 17, 2025
*/
#include "Human.h"
#include <iostream>

int main()
{
    double x = 2000, y = 3000;
    Human h("Ken Lai", x, y);
    h.display();

    return 0;
}
```

```
/*
*
*
File Name: point.cpp
Assignment: Lab 1 Exercise D
*  Completed by: John Zhou
*  Submission Date: Sept 17, 2025
*/
#include "Point.h"


Point::Point(double a, double b) : x(a), y(b) {}
double Point::get_x() const { return x; }
double Point::get_y() const { return y; }
void Point::set_x(double a) { x = a; }
void Point::set_y(double a) { y = a; }
```

```
/*
*
*
File Name: point.h
Assignment: Lab 1 Exercise D
*  Completed by: John Zhou
*  Submission Date: Sept 17, 2025
*/
#ifndef POINT_H
#define POINT_H

class Point
{
private:
    double x;
    double y;

public:
    Point(double a = 0, double b = 0);
    double get_x() const;
    double get_y() const;
    void set_x(double a);
    void set_y(double a);
};

#endif
```

Screenshot



```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/614/ENSF-614-assignment-
repo/assignment1/exerciseD
$ ./ed.exe
Human Name: Ken Lai
Human Location: 2000 ,3000.
```