Course: Programming Fundamental - ENSF 480
Lab #: Lab 1
Instructor: M. Moussavi
Student Name: Daniel Rey, Aly Farouz
Lab Section: B01
Date Submitted: September 15, 2025

Exercise B

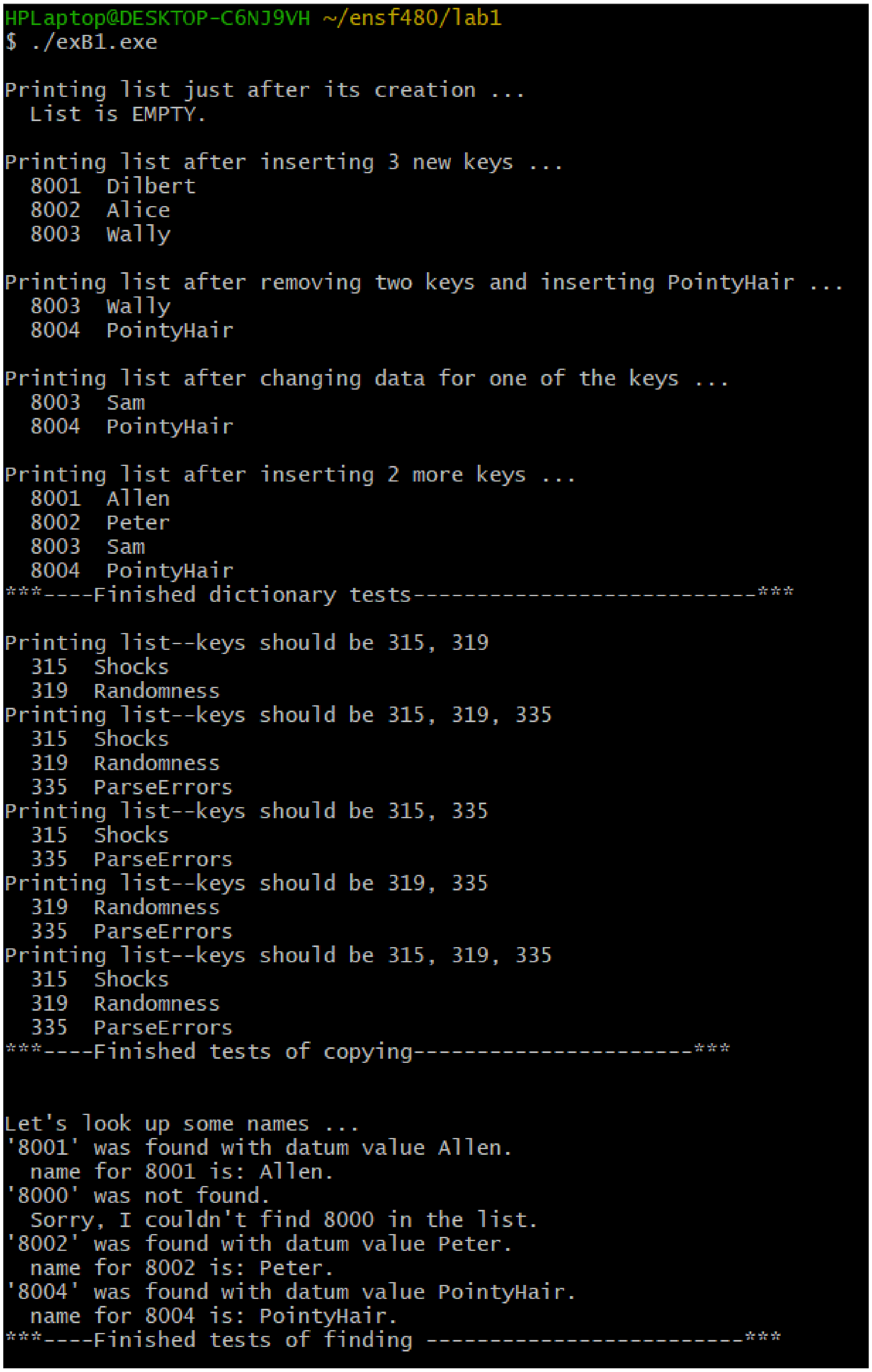
```
/*
* File Name: dictionaryList.cpp
* Assignment: Lab 1 Exercise B
* Lab section: B01
* Functions modified by: Daniel Rey, Aly Farouz
  - DictionaryList(const DictionaryList& source)
  - operator =(const DictionaryList& rhs)
  - ~DictionaryList()
  - make_empty()
* Submission Date: Sept 15, 2025
*/


#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"

using namespace std;

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

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

DictionaryList::DictionaryList(const DictionaryList& source)
  : sizeM(source.sizeM), headM(0), cursorM(source.cursorM)
{
  if (sizeM!=0)
  {
    Node *c = source.headM;
    Node *p, *prev = headM = new Node(c->keyM, c->datumM, NULL);
    for(c = c->nextM; c!=NULL; c = c->nextM)
    {
      p = new Node(c->keyM, c->datumM, NULL);
      prev = prev->nextM = p;
    }
  }
}
```

```
DictionaryList& DictionaryList::operator =(const DictionaryList& rhs)
{
  if (this != &rhs) {
    if (sizeM>=rhs.sizeM)
    {
      if (headM!=NULL)
      {
        Node *c, *p = headM;
        for(c=rhs.headM; c!=NULL; p=p->nextM)
        {
          p->keyM = c->keyM;
          p->datumM = c->datumM;
          c = c->nextM;
        }
        Node *next = p->nextM;
        p->nextM = NULL;
        while(next!=NULL)
        {
          p = next;
          next = next->nextM;
          delete p;
        }
      }
    }
    else
    {
      Node *prev, *p, *c = rhs.headM;
      if (headM==NULL)
      {
        prev = headM = new Node(c->keyM, c->datumM, NULL);
        c = c->nextM;
      }
      else
      {
        for(p=headM; p!=NULL; p=p->nextM)
        {
          p->keyM = c->keyM;
          p->datumM = c->datumM;
          c = c->nextM;
          prev = p;
        }
      }
      while(c!=NULL)
      {
        p = new Node(c->keyM, c->datumM, NULL);
        prev = prev->nextM = p;
        c = c->nextM;
      }
    }
    cursorM = rhs.cursorM;
    sizeM = rhs.sizeM;
  }
  return *this;
}
```

```
DictionaryList::~DictionaryList()
{
  for(Node *prev, *p=headM; p!=NULL;)
  {
    prev = p;
    p = p->nextM;
    delete prev;
  }
}

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

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

const Key& 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;
}
```

```
void DictionaryList::make_empty()
{
  for(Node *prev, *p=headM; p!=NULL;)
  {
    prev = p;
    p = p->nextM;
    delete prev;
  }
  cursorM = headM = 0;
  sizeM = 0;
}
```

```
void DictionaryList::find(const Key& keyA)
{
    for (Node *p = headM; p != 0; p=p->nextM)
        if (keyA == p->keyM)
        {
            cout << "'" << keyA <<"' was found with datum value " << p->datumM.c_str() <<
".\n";
            cursorM = p;
            return;
        }
    cout << "'" << keyA <<"' was not found.\n";
    cursorM = 0;
}


void DictionaryList::destroy()
{

        Node *p = headM;
        Node *prev;
        while (p != 0)
        {
            prev = p;
            p = p->nextM;
            delete prev;
        }
        headM = 0;
        sizeM = 0;

}


void DictionaryList::copy(const DictionaryList& source)
{
    if (source.headM == 0) {
        headM = 0;
        return;
    }

    headM = new Node (source.headM->keyM, source.headM->datumM, NULL);
    Node *newest_node = headM;

    const Node *source_node = source.headM;

    if(source_node == source.cursorM)
         cursorM = newest_node;

    while (true) {
        source_node = source_node->nextM;

        if (source_node == 0)
            break;

        newest_node->nextM = new Node(source_node->keyM, source_node->datumM, NULL);

       if(source_node == source.cursorM)
            cursorM = newest_node->nextM;

        newest_node = newest_node->nextM;

    }

    sizeM = source.sizeM;

}
```



```
HPLaptop@DESKTOP-C6NJ9VH ~/ensf480/lab1
$ ./exB1.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 ...
'8001' was found with datum value Allen.
  name for 8001 is: Allen.
'8000' was not found.
  Sorry, I couldn't find 8000 in the list.
'8002' was found with datum value Peter.
  name for 8002 is: Peter.
'8004' was found with datum value PointyHair.
  name for 8004 is: PointyHair.
***----Finished tests of finding -------------------------***
```

Exercise C

```
/*
* File Name: company.cpp
* Assignment: Lab 1 Exercise C
* Lab section: B01
* Completed by: Daniel Rey, Aly Farouz
* Submission Date: Sept 15, 2025
*/
#include <string>
#include <vector>
using namespace std;


class Person {
friend class Company;
protected:
  string nameM;
  string phoneM;
  string addressM;
};

class Employee: public Person {
friend class Company;
private:
  const string dateOfBirthM;      // employee's birth date (YYYY/MM/DD)
  string stateM;                  // (active, suspended, retired, fired)
};

class Company {
private:
  string nameM;                   // company's name
  string addressM;                // company's address
  const string foundingDateM;     // the date that company was established

  vector<Employee> employeesM;    // vector of employees

  vector <Person> customersM;     // vector of customers

};
```

Exercise D

```
/*
* File Name: human.h
* Assignment: Lab 1 Exercise D
* Lab section: B01
* Completed by: Daniel Rey, Aly Farouz
* Submission Date: Sept 15, 2025
*/

#ifndef HUMAN_H
#define HUMAN_H
#include <cstring>
#include <iostream>

using namespace std;
class Point{
private:
  double x;       // x coordinate of a location on Cartesian Plain
  double y;       // y coordinate of a location on Cartesian Plain
public:
  Point(double a =0, double b =0);

  double get_x() const;
  //PROMISSES: Returns the x coordinate.

  double get_y() const;
  //PROMISSES: Returns the y coordinate.

  void set_x(double a);
  //PROMISSES: Sets the x coordinate to a.

  void set_y(double a);
  //PROMISSES: Sets the y coordinate to a.
};

class Human {
private:
  Point location;   // Location of an object of Human on a Cartesian Plain
  char *name;       // Human's name
public:
  Human(const char* nam="", double x=0, double y=0);
  ~Human();

  const char* get_name() const;
  //PROMISSES: Returns the Human's name

  void set_name(char* name);
  //PROMISSES: Human's name is changed to the name passed to this function

  Point get_point() const;
  //PROMISSES: Returns the Human's location as a Point object

  void set_point(double x, double y);
  //PROMISSES: Human's location coordinates are changed to x and y.

  void display() const;
  //PROMISSES: Prints the information about the Human to the output stream
};

#endif
```

```cpp
/*
* File Name: human.cpp
* Assignment: Lab 1 Exercise D
* Lab section: B01
* Completed by: Daniel Rey, Aly Farouz
* Submission Date: Sept 15, 2025
*/

#include <cstring>
#include <iostream>
#include "human.h"
using namespace std;


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;};


Human::Human(const char* nam, double x, double y): name(new char[strlen(nam)+1]) {
  strcpy(this->name, nam);
  location.set_x(x);
  location.set_y(y);
}

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

const char* Human::get_name()const {return name;}
void Human::set_name(char* name) {
  delete [] this->name;
  this->name = new char[strlen(name)+1];
  strcpy(this->name, name);
}

Point Human::get_point()const {return location;}
void Human::set_point(double x, double y) {
  location.set_x(x);
  location.set_y(y);
}

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

```
/*
* File Name: exDmain.cpp
* Assignment: Lab 1 Exercise D
* Lab section: B01
* Completed by: Daniel Rey, Aly Farouz
* Submission Date: Sept 15, 2025
*/

#include <iostream>
#include "human.h"
using namespace std;


int main(int argc, char **argv)
{
  double x = 2000, y = 3000;
  Human h("Ken Lai", x , y);
  h.display();
  return 0;
}
```