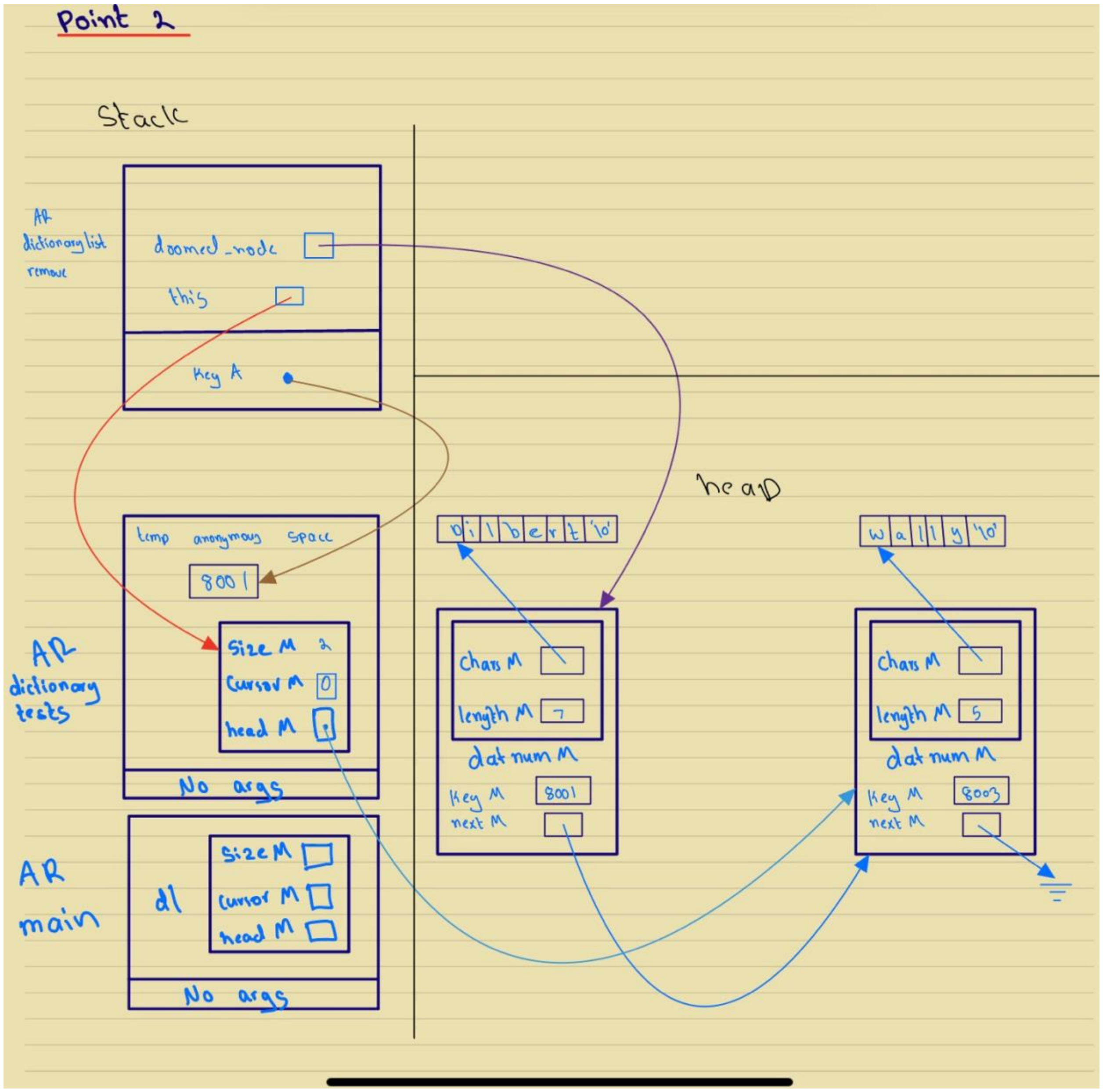
Name	**: Nimna Wijedasa**
Course Name	**: Principles of Software Design**
Lab Section	**: B02**
Course Code	**: ENSF 480**
Assignment Number	**: Lab-1**
Submission Date	**: 21/09/2023**

Exercise A

Program output and its order	Your explanation (why and where is the cause for this output)
constructor with int argument is called.	Called in line 12 of exAmain “Mystring c = 3” calls the constructor Mystring::Mystring(int n):
default constructor is called. **default constructor is called.**	Called in line 18 of exAmain “Mystring x[2];” calls the default constructor “Mystring::Mystring()” twice
constructor with char* argument is called.	Called in line 22 of exAmain “Mystring("4")” calls the constructor with the char* argument “Mystring::Mystring(const char *s);”
copy constructor is called. **copy constructor is called.**	Called in line 24 of exAmain “x[0].append(*z).append(x[1]);” calls the copy constructor “Mystring::Mystring(const Mystring& source):” twice
destructor is called. **destructor is called.**	Called in line 24 of exAmain “x[0].append(*z).append(x[1]);” it is called twice as the append function terminates.
copy constructor is called.	Called in line 26 of exAmain “Mystring mars = x[0];” it calls the copy constructor “Mystring::Mystring(const Mystring& source):”
assignment operator called.	Called in line 28 of exAmain “x[1] = x[0]; it calls the assignment operator “Mystring& Mystring::operator =(const Mystring& S)”
constructor with char* argument is called. **constructor with char* argument is called.**	Called in line 30 and 32 of exAmain “ Mystring jupiter("White");” and “ar[0] = new Mystring ("Yellow");” . It calls the constructor with the char* argument “Mystring::Mystring(const char *s);” both times
destructor is called. **destructor is called.** **destructor is called.** **destructor is called.** **destructor is called.**	Called 4 times after line 34 of exAmain when the block ends. Because it’s called twice for “Mystring x[2];” and once for “Mystring jupiter("White");” and once for ”ar[0] = new Mystring ("Yellow");”. The last call to the destructor is from “ delete ar [0];” on line 37.
constructor with char* argument is called.	Called in line 39 of exAmain “ Mystring d = "Green";” calls the constructor with the char* argument “Mystring::Mystring(const char *s);”
Program terminated successfully.	Called in line 41 of exAmain, it’s the cout of the function to indicate the end of main
destructor is called. **destructor is called**	Called after the “return 0” in line 42 of exAmain it destroys the created strings “ Mystring d = "Green";” and “ Mystring c = 3;” by calling the “Mystring::~Mystring()” as the variables go out of scope.

Exercise B


Point 2
Stack
AR dictionary list remove
doomed_node
this
Key A
head
temp anonymous space
8001
AR dictionary tests
Size M 2
Cursor M 0
head M
No args
AR main
dl
Size M
cursor M
head M
No args
D i l b e r t '\0'
Chars M
length M 7
dat num M
Key M 8001
next M
W a l l y '\0'
Chars M
length M 5
dat num M
Key M 8003
next M

```cpp
//  File Name                          : dictionaryList.cpp
//  Assignment and exercise number     : Assignment 1 Exercise B
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"
#include "mystring_B.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)
{
  copy(source);
}

DictionaryList& DictionaryList::operator =(const DictionaryList& rhs)
{
  if (this != &rhs) {
    destroy();
    copy(rhs);
  }
  return *this;
}
```

```
DictionaryList::~DictionaryList()
{
  destroy();
}

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 Mystring& 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()
{
    destroy();
    sizeM = 0;
    cursorM = 0;
}


// The following function are supposed to be completed by the stuents, as part
// of the exercise B part II. the given fucntion are in fact place-holders for
// find, destroy and copy, in order to allow successful linking when you're
// testing insert and remove. Replace them with the definitions that work.

void DictionaryList::find(const Key& keyA)
{
  if(headM == 0 || keyA < headM -> keyM){

    cursorM = 0;
    return;
  }

  else if(keyA == headM->keyM)
    cursorM = headM;

  else{
    Node *p = headM ->nextM;
    Node *prev = headM;

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

```
    prev = p;
    p = p->nextM;
   }
  if(p != 0 && p->keyM == keyA)
   cursorM = p;
  else
   cursorM = 0;
 }

}


void DictionaryList::destroy()
{
 Node *p = headM;
 while (p != 0) {
  Node *next = p->nextM;
  delete p;
  p = next;
 }
 headM = 0;
 cursorM = 0;
}


void DictionaryList::copy(const DictionaryList& source)
{

 if (source.headM == 0) {
  headM = 0;
  cursorM = 0;
  sizeM = 0;
 }
 else {
  Node *tail_copy = new Node(source.headM->keyM, source.headM->datumM, 0);
  headM = tail_copy;
  sizeM = 1;
```

```
    if (source.cursorM == source.headM)
      cursorM = headM;
    for (Node *p = source.headM->nextM; p != 0; p = p->nextM) {
      tail_copy->nextM = new Node(p->keyM, p->datumM, 0);
      tail_copy = tail_copy->nextM;
      sizeM++;
      if (p == source.cursorM)
        cursorM = tail_copy;
    }
  }
}
```

```
//  File Name                          : dictionaryList.h
//  Assignment and exercise number    : Assignment 1 Exercise B
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023

#ifndef DICTIONARY_H
#define DICTIONARY_H
#include <iostream>
using namespace std;

// class DictionaryList: GENERAL CONCEPTS
//
//    key/datum pairs are ordered.  The first pair is the pair with
//    the lowest key, the second pair is the pair with the second
//    lowest key, and so on.  This implies that you must be able to
//    compare two keys with the < operator.
//
//    Each DictionaryList object has a "cursor" that is either attached
//    to a particular key/datum pair or is in an "off-list" state, not
//    attached to any key/datum pair.  If a DictionaryList is empty, the
//    cursor is automatically in the "off-list" state.

#include "mystring_B.h"

// Edit these typedefs to change the key or datum types, if necessary.
typedef int Key;
typedef Mystring Datum;

// THE NODE TYPE
//    In this exercise the node type is a class, that has a ctor.
//    Data members of Node are private, and class DictionaryList
//    is declared as a friend. For details on the friend keyword refer to your
//    lecture notes.

class Node {
  friend class DictionaryList;
private:
  Key keyM;
  Datum datumM;
  Node *nextM;

  // This ctor should be convenient in insert and copy operations.
  Node(const Key& keyA, const Datum& datumA, Node *nextA);
};

class DictionaryList {
public:
  DictionaryList();
  DictionaryList(const DictionaryList& source);
  DictionaryList& operator =(const DictionaryList& rhs);
  ~DictionaryList();

  int size() const;
  // PROMISES: Returns number of keys in the table.

  int cursor_ok() const;
```

```
// PROMISES:
//   Returns 1 if the cursor is attached to a key/datum pair,
//   and 0 if the cursor is in the off-list state.

const Key& cursor_key() const;
// REQUIRES: cursor_ok()
// PROMISES: Returns key of key/datum pair to which cursor is attached.

const Datum& cursor_datum() const;
// REQUIRES: cursor_ok()
// PROMISES: Returns datum of key/datum pair to which cursor is attached.

void insert(const Key& keyA, const Datum& datumA);
// PROMISES:
//   If keyA matches a key in the table, the datum for that
//   key is set equal to datumA.
//   If keyA does not match an existing key, keyA and datumM are
//   used to create a new key/datum pair in the table.
//   In either case, the cursor goes to the off-list state.

void remove(const Key& keyA);
// PROMISES:
//   If keyA matches a key in the table, the corresponding
//   key/datum pair is removed from the table.
//   If keyA does not match an existing key, the table is unchanged.
//   In either case, the cursor goes to the off-list state.

void find(const Key& keyA);
// PROMISES:
//   If keyA matches a key in the table, the cursor is attached
//   to the corresponding key/datum pair.
//   If keyA does not match an existing key, the cursor is put in
//   the off-list state.

void go_to_first();
// PROMISES: If size() > 0, cursor is moved to the first key/datum pair
//   in the table.
```

```
  void step_fwd();
  // REQUIRES: cursor_ok()
  // PROMISES:
  //   If cursor is at the last key/datum pair in the list, cursor
  //   goes to the off-list state.
  //   Otherwise the cursor moves forward from one pair to the next.

  void make_empty();
  // PROMISES: size() == 0.

private:
  int sizeM;
  Node *headM;
  Node *cursorM;

  void destroy();
  // Deallocate all nodes, set headM to zero.

  void copy(const DictionaryList& source);
  // Establishes *this as a copy of source.  Cursor of *this will
  // point to the twin of whatever the source's cursor points to.

};

#endif
```

```cpp
//  File Name                          : exBmain.cpp
//  Assignment and exercise number    : Assignment 1 Exercise B
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023
#include <assert.h>
#include <iostream>
#include "dictionaryList.h"

using namespace std;

DictionaryList dictionary_tests();

void test_copying();

void print(DictionaryList& dl);

void test_finding(DictionaryList& dl);

void test_operator_overloading(DictionaryList& dl);

int main()
{
  DictionaryList dl = dictionary_tests();

  test_copying();

 // Uncomment the call to test_copying when DictionaryList::copy is properly defined
 test_finding(dl);
 // test_operator_overloading(dl);

  return 0;
}

DictionaryList dictionary_tests()
{

  DictionaryList dl;

  assert(dl.size() == 0);
  cout << "\nPrinting list just after its creation ...\n";
  print(dl);

  // Insert using new keys.
  dl.insert(8001,"Dilbert");
  dl.insert(8002,"Alice");
  dl.insert(8003,"Wally");
  assert(dl.size() == 3);
  cout << "\nPrinting list after inserting 3 new keys ...\n";
  print(dl);
  dl.remove(8002);
  dl.remove(8001);
  dl.insert(8004,"PointyHair");
  assert(dl.size() == 2);
  cout << "\nPrinting list after removing two keys and inserting PointyHair ...\n";
  print(dl);

  // Insert using existing key.
  dl.insert(8003,"Sam");
  assert(dl.size() == 2);
  cout << " \nPrinting list after changing data for one of the keys ... \n";
  print(dl);

  dl.insert(8001,"Allen");
  dl.insert(8002,"Peter");
  assert(dl.size() == 4);
  cout << " \nPrinting list after inserting 2 more keys ... \n";
  print(dl);

  cout << "***----Finished dictionary tests---------------------------*** \n\n";
  return dl;
}

void test_copying()
{
   DictionaryList one;

  // Copy an empty list.
  DictionaryList two;
  assert(two.size() == 0);

  // Copy a list with three entries and a valid cursor.
  one.insert(319,"Randomness");
  one.insert(315,"Shocks");
  one.insert(335,"ParseErrors");
  one.go_to_first();
  one.step_fwd();

  DictionaryList three(one);

  assert(three.cursor_datum().isEqual("Randomness"));
  one.remove(335);

  cout << "Printing list--keys should be 315, 319 \n";
  print(one);

  cout << "Printing list--keys should be 315, 319, 335 \n";
  print(three);

  // Assignment operator check.
  one = two = three = three;
  one.remove(319);
  two.remove(315);

  cout << "Printing list--keys should be 315, 335\n";
  print(one);

  cout << "Printing list--keys should be 319, 335\n";
  print(two);

  cout << "Printing list--keys should be 315, 319, 335\n";
  print(three);

  cout << "***----Finished tests of copying----------------------***\n\n";
}

void print(DictionaryList& dl)
{
  if (dl.size() == 0)
    cout << "  List is EMPTY.\n";
  for (dl.go_to_first(); dl.cursor_ok(); dl.step_fwd()) {
    cout << "  " << dl.cursor_key();
    cout << "  " << dl.cursor_datum().c_str() << '\n';
  }
}

void test_finding(DictionaryList& dl)
{

    // Pretend that a user is trying to look up names.
    cout << "\nLet's look up some names ...\n";

    dl.find(8001);
    if (dl.cursor_ok())
```

```
      cout << "  name for 8001 is: " << dl.cursor_datum().c_str() << ". \n";
    else
      cout << "  Sorry, I couldn't find 8001 in the list.  \n" ;

    dl.find(8000);
    if (dl.cursor_ok())
      cout << "  name for 8000 is: " << dl.cursor_datum().c_str() << ". \n";
    else
      cout << "  Sorry, I couldn't find 8000 in the list.  \n" ;

    dl.find(8002);
    if (dl.cursor_ok())
      cout << "  name for 8002 is: " << dl.cursor_datum().c_str() << ". \n";
    else
      cout << "  Sorry, I couldn't find 8002 in the list.  \n" ;

    dl.find(8004);
    if (dl.cursor_ok())
      cout << "  name for 8004 is: " << dl.cursor_datum().c_str() << ". \n";
    else
      cout << "  Sorry, I couldn't find 8004 in the list.  \n" ;

   cout << "***----Finished tests of finding -------------------------*** \n\n";
}
#if 0
void test_operator_overloading(DictionaryList& dl)
{

   DictionaryList dl2 = dl;
   dl.go_to_first();
   dl.step_fwd();
   dl2.go_to_first();

   cout << " \nTestig a few comparison and insertion operators." << endl;

   // Needs to overload >= and << (insertion operator) in class Mystring
   if(dl.cursor_datum() >= (dl2.cursor_datum()))
```

```
     cout << endl << dl.cursor_datum() << " is greater than or equal " << dl2.cursor_datum();
  else
     cout << endl << dl2.cursor_datum() << " is greater than " << dl.cursor_datum();

  // Needs to overload <= for Mystring
  if(dl.cursor_datum() <= (dl2.cursor_datum()))
     cout << dl.cursor_datum() << " is less than or equal" << dl2.cursor_datum();
  else
     cout << endl << dl2.cursor_datum() << " is less than " << dl.cursor_datum();

  if(dl.cursor_datum() != (dl2.cursor_datum()))
     cout << endl << dl.cursor_datum() << " is not equal to " << dl2.cursor_datum();
  else
     cout << endl << dl2.cursor_datum() << " is equal to " << dl.cursor_datum();



  if(dl.cursor_datum() > (dl2.cursor_datum()))
     cout << endl << dl.cursor_datum() << " is greater than " << dl2.cursor_datum();
  else
     cout << endl << dl.cursor_datum() << " is not greater than " << dl2.cursor_datum();

  if(dl.cursor_datum() < (dl2.cursor_datum()))
     cout << endl << dl.cursor_datum() << " is less than " << dl2.cursor_datum();
  else
     cout << endl << dl.cursor_datum() << " is not less than " << dl2.cursor_datum();
  if(dl.cursor_datum() == (dl2.cursor_datum()))
     cout << endl << dl.cursor_datum() << " is equal to " << dl2.cursor_datum();
  else
     cout << endl << dl.cursor_datum() << " is not equal to " << dl2.cursor_datum();
 cout << endl << "\nUsing square bracket [] to access elements of Mystring objects. ";

  char c = dl.cursor_datum()[ 1];
  cout << endl << "The socond element of "  << dl.cursor_datum() << " is: " << c;

  dl.cursor_datum()[ 1] = 'o';
  c = dl.cursor_datum()[ 1];
  cout << endl << "The socond element of "  << dl.cursor_datum() << " is: " << c;
```

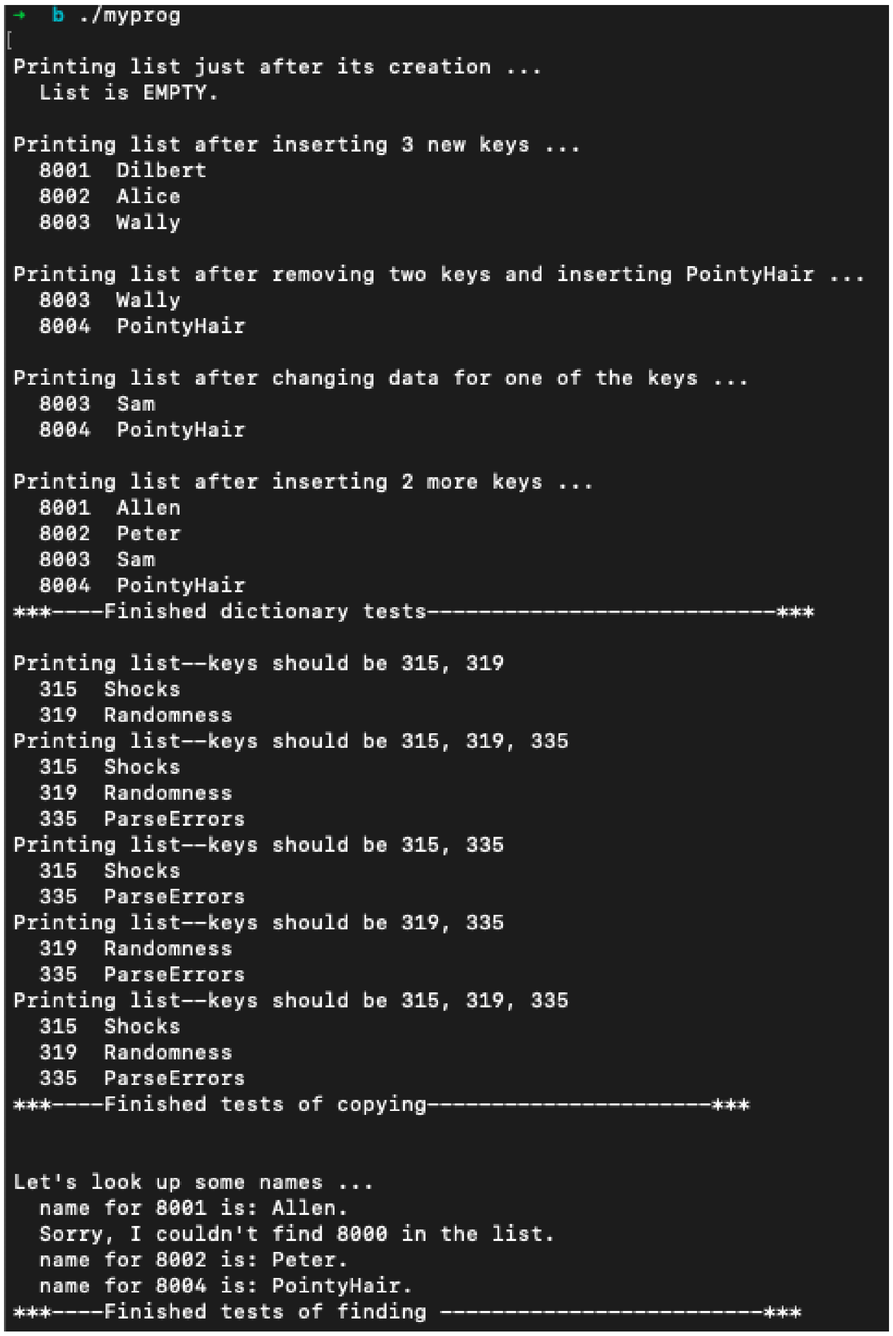
```
cout << endl << "\nUsing << to display key/datum pairs in a Dictionary list: \n";
/* The following line is expected to display the content of the linked list
 * dl2 -- key/datum pairs. It should display:
 *   8001  Allen
 *   8002  Peter
 *   8003  Sam
 *   8004  PointyHair
 */
cout << dl2;

cout << endl << "\nUsing [] to display the datum only: \n";
/* The following line is expected to display the content of the linked list
 * dl2 -- datum. It should display:
 *   Allen
 *   Peter
 *   Sam
 *   PointyHair
 */

for(int i = 0; i < dl2.size(); i++)
   cout << dl2[i] << endl;

cout << endl << "\nUsing [] to display sequence of charaters in a datum: \n";
/* The following line is expected to display the characters in the first node
 * of the dictionary. It should display:
 *   A
 *   l
 *   l
 *   e
 *   n
 */
cout << dl2[0][0] << endl;
cout << dl2[0][1] << endl;
cout << dl2[0][2] << endl;
cout << dl2[0][3] << endl;
cout << dl2[0][4] << endl;
```

```
    cout << "\n\n***----Finished tests for overloading operators ----------*** \n\n";
}
#endif
```



```
➜  b ./myprog
[
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 C

```
//  File Name                          : company.h
//  Assignment and exercise number    : Assignment 1 Exercise C
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023

#ifndef COMPANY_H
#define COMPANY_H

#include <string>
#include <vector>

class EmployeeManager;

class Company {
private:
    std::string companyName;
    std::string companyAddress;
    struct DateEstablished {
        int day;
        int month;
        int year;
    } establishmentDate;

    std::vector<std::string> customers;

    friend class EmployeeManager;

public:
    Company(const std::string& name, const std::string& address, int day, int month, int year);

    std::string getCompanyName() const;
    std::string getCompanyAddress() const;
    DateEstablished getDateEstablished() const;
    void addCustomer(const std::string& customerInfo);
};

#endif
```

```
// File Name                    : company.cpp
// Assignment and exercise number    : Assignment 1 Exercise C
// Lab section                  : B02
// Your name                    : Nimna Wijedasa
// Submission Date              : Sept 18, 2023
#include "Company.h"

Company::Company(const std::string& name, const std::string& address, int day, int month, int year)
    : companyName(name), companyAddress(address) {
    establishmentDate.day = day;
    establishmentDate.month = month;
    establishmentDate.year = year;
}

std::string Company::getCompanyName() const {
    return companyName;
}

std::string Company::getCompanyAddress() const {
    return companyAddress;
}

Company::DateEstablished Company::getDateEstablished() const {
    return establishmentDate;
}

void Company::addCustomer(const std::string& customerInfo) {
    customers.push_back(customerInfo);
}
```

```cpp
// File Name                        : EmployeeManager.h
// Assignment and exercise number   : Assignment 1 Exercise C
// Lab section                      : B02
// Your name                        : Nimna Wijedasa
// Submission Date                  : Sept 18, 2023
#ifndef EMPLOYEEMANAGER_H
#define EMPLOYEEMANAGER_H

#include <vector>
#include <string>

class Company;

class EmployeeManager {
private:
    std::vector<std::string> employees;
    std::vector<std::string> employeeState;

public:
    void addEmployee(Company& company, const std::string& employeeInfo);
    void addEmployeeState(Company& company, const std::string& state);
};

#endif
```

```cpp
// File Name                        : EmployeeManager.cpp
// Assignment and exercise number   : Assignment 1 Exercise C
// Lab section                      : B02
// Your name                        : Nimna Wijedasa
//  Submission Date                  : Sept 18, 2023
#include "EmployeeManager.h"
#include "Company.h"

void EmployeeManager::addEmployee(Company& company, const std::string& employeeInfo) {
    company.employees.push_back(employeeInfo);
}

void EmployeeManager::addEmployeeState(Company& company, const std::string& state) {
    company.employeeState.push_back(state);
}
```

Exercise D

```
//  File Name                        : human.h
//  Assignment and exercise number    : Assignment 1 Exercise D
//  Lab section                      : B02
//  Your name                        : Nimna Wijedasa
//  Submission Date                  : Sept 18, 2023
#ifndef HUMAN_H
#define HUMAN_H
#include "point.h"
#include <string>

class Human {
```

```
private:
    Point location;
    std::string name;

public:
    Human(const std::string& nam = "", double x = 0, double y = 0);
    std::string get_name() const;
    void set_name(const std::string& nam);
    Point get_point() const;
    virtual void display() const;
    ~Human();
};

#endif
```

```
//  File Name                          : human.cpp
//  Assignment and exercise number    : Assignment 1 Exercise D
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023
#include "Human.h"
#include <iostream>

Human::Human(const std::string& nam, double x, double y) : name(nam), location(x, y) {}

std::string Human::get_name() const {
    return name;
}

void Human::set_name(const std::string& nam) {
    name = nam;
}

Point Human::get_point() const {
    return location;
}
```

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

Human::~Human() {
}
```

```
//  File Name                          : point.h
//  Assignment and exercise number    : Assignment 1 Exercise D
//  Lab section                        : B02
//  Your name                          : Nimna Wijedasa
//  Submission Date                    : Sept 18, 2023
#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 b);
};

#endif
```

```
//  File Name                          : point.cpp
```

```cpp
//  Assignment and exercise number    : Assignment 1 Exercise D
//  Lab section                       : B02
//  Your name                         : Nimna Wijedasa
//  Submission Date                   : Sept 18, 2023
#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 b) {
    y = b;
}
```

```cpp
//  File Name                         : main.cpp
//  Assignment and exercise number    : Assignment 1 Exercise D
//  Lab section                       : B02
//  Your name                         : Nimna Wijedasa
//  Submission Date                   : Sept 18, 2023
#include "Human.h"

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