ENSF 480 Lab 1 Report

Cover Page

Names: Beau McCartney, Apostolos Scondriannis

Course Name: Principles of Software Design

Lab Section: B02 (Dr. Moshirpour)

Course Code: ENSF 480 Fall 2021

Assignment Number: Lab 1

Submission Date and Time: 24/09/2021

Exercise Solutions

Exercise A

Answer the questions in Exercise A in the following table and post it into the D2L:

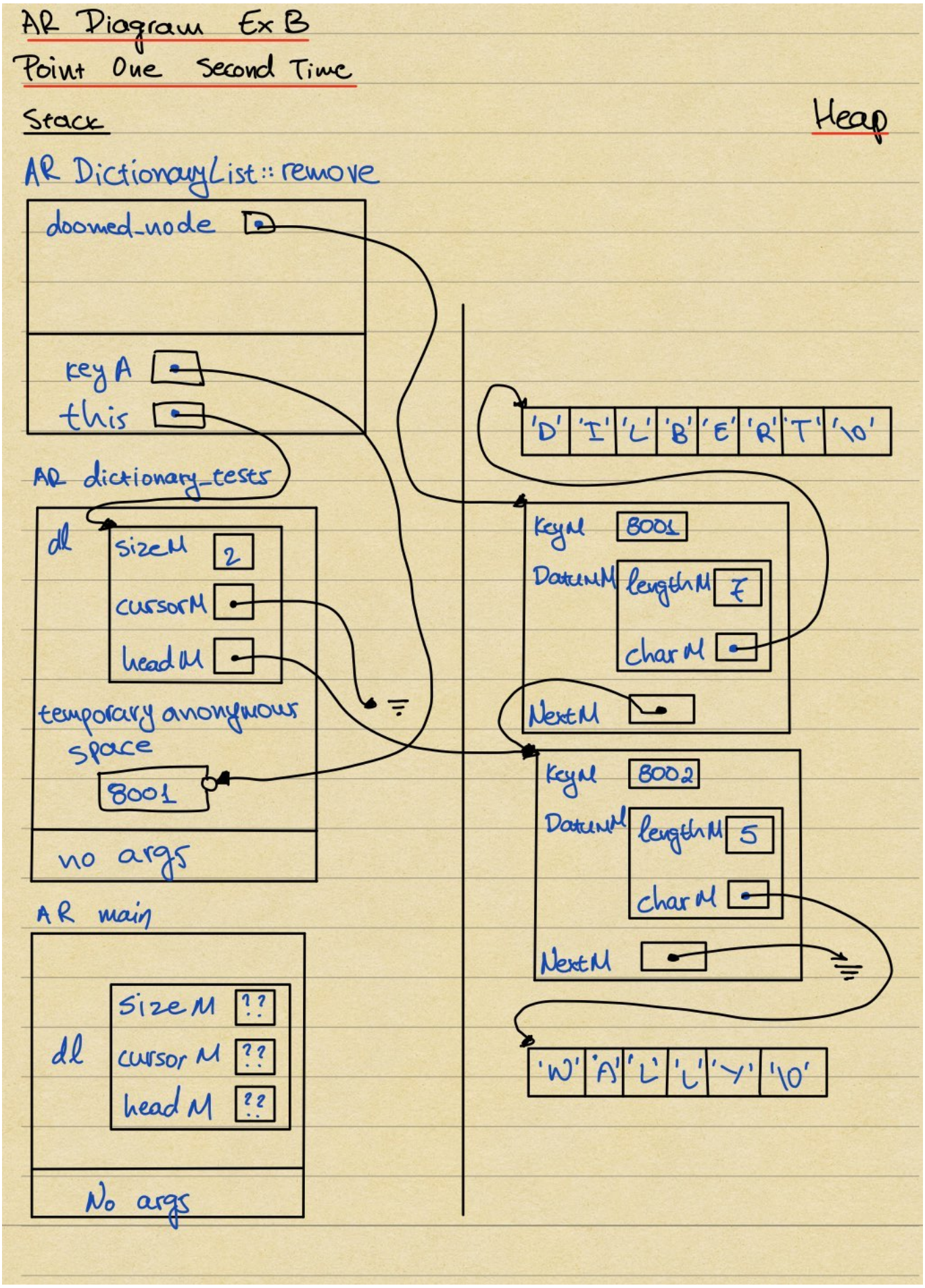
Program output and its order	Your explanation (why and where is the cause for this output)
constructor with int argument is called.	It is called at line 12 in exAmain. The statement "Mystring c = 3;" is interpreted by the compiler as a call to the constructor Mystring::Mystring(int n).
default constructor is called. default constructor is called.	It is called at line 18 of exAmain. The statement "Mystring x[2]" creates an array of two Mystring objects, which in turn calls the default constructor to make those two Mystring objects
constructor with char* argument is called.	It is called at line 22 of exAmain. The statement "Mystring* z = new Mystring("4");" allocates a new Mystring object on the heap. "4" is a string literal, interpreted by the compiler as a character array (i.e. pointer) that points to the head of the array |'4'|'\0'|, therefore the constructor that fits the arguments is the one accepting the char pointer, as the argument is read as a char pointer.
copy constructor is called. copy constructor is called.	These are both called at line 24 of exAmain. The statement "x[0].append(*z).append(x[1]);" calls the Mystring::append(const Mystring other) function, which accepts an argument of type Mystring. Because the argument's type is not Mystring* or Mystring&, and a copy constructor is defined, the argument is passed by value. This means it must be copied into the parameter "other", which is done using the copy constructor. Two calls to append() means two arguments copied
destructor is called. destructor is called.	The first and second "destructor is called." output is created at the line 25. It is called once at the end of each append call for the object MyString "other".
copy constructor is called.	Called at line 25 of exAmain. The statement "Mystring mars = x[0];" is interpreted by the compiler as a call to the constructor Mystring::Mystring(const Mystring& source), where "source" is the Mystring object x[0].
assignment operator called.	At line 28 of exAmain.cpp. The compiler interprets "x[1] = x[0];" as a call to Mystring& Mystring::operator =(const Mystring& S) from x[1], with "S" being x[0]

constructor with char* argument is called.
constructor with char* argument is called.
	// Mystring jupiter("White");
	
	// ar[0] = new Mystring ("Yellow");

destructor is called. destructor is called. destructor is called.	The following objects go out of scope at line 34 of exAmain: x[0], x[1], *z, mars, and jupiter. To free their

destructor is called. **destructor is called.**	memory as they go out of scope, there is a destructor call for each object. 5 objects, 5 destructor calls
constructor with char* argument is called.	It is called at line 39 in exAmain. The statement "Mystring d = "Green";" is interpreted by the compiler as a call to the constructor Mystring::Mystring(const char *s).
Program terminated successfully.	We've reached the end of main, just before the return without any errors. There, there is a cout statement explicitly calling for this string to be printed.
destructor is called. **destructor is called**	c and d go out of scope, meaning that they are destructed. Note that *ar[1] is a nullptr, *ar[0] was destructed at the end of the last scope, and *ar[2] is a reference to c (which is already being destructed, and can't be again), so none of the elements in ar can actually be destructed. The only other objects in scope are c and d. 2 objects, 2 destructor calls

Exercise B


AR Diagram Ex B
Point One Second Time
Stack
Heap
AR DictionaryList::remove
doomed_node
keyA
this
AR dictionary_tests
dl
sizeM
2
cursorM
headM
temporary anonymous space
8001
no args
AR main
dl
sizeM
??
cursorM
??
headM
??
No args
'D' 'I' 'C' 'B' 'E' 'R' 'T' '\0'
keyM
8001
DatumM
lengthM
7
charM
NextM
keyM
8002
DatumM
lengthM
5
charM
NextM
'W' 'A' 'L' 'L' 'Y' '\0'

Code

```
/*
 * File Name: dictionaryList.cpp
 * Assignment: Lab 1 Exercise B
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#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 (sizeM == 0) {
        cursorM = 0;
        return;
    }

    Node *p;
    for(p = headM; p != 0 && p -> keyM < keyA; p = p -> nextM)
        ;

    cursorM = p != 0 && p -> keyM == keyA ? p : 0;
}


void DictionaryList::destroy()
{
  if (sizeM > 0) {
      for (Node *temp = headM->nextM; temp != 0; temp = temp -> nextM) {
          delete headM;
          headM = temp;
      }
      delete headM;

  }
  sizeM = 0;
  headM = 0;
  cursorM = 0;
}


void DictionaryList::copy(const DictionaryList& source)
{
    sizeM = 0;
    headM = 0;

    Node *src = source.headM;

    while (src != 0)
    {
        const Key keyN = src->keyM;
        const Datum datumN = src->datumM;
        insert(keyN, datumN);
```

```
            src = src->nextM;
        }

        if (source.cursor_ok()) {
            Key cursorKey = source.cursor_key();
            find(cursorKey);
        }

        assert(sizeM == source.size());
    }
```

```
✝ ENSF480/Labs/ENSF480-Lab_1  main± g++ -Wall -g -ggdb exB/*.cpp -o bin/a.out
✝ ENSF480/Labs/ENSF480-Lab_1  main± ./bin/a.out

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

Code

```
/*
 * File Name: Company.h
 * Assignment: Lab 1 Exercise C
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#ifndef COMPANY_H
#define COMPANY_H
#include <string>
#include <vector>
#include "Employee.h"
#include "Customer.h"
using namespace std;

// we are trying to refactor. Let's use classes instead right??
class Company {
    private:
        string name;
        string address;
        vector<Employee> employees;
        string dateEstablished;
        vector<Customer> customers;
};

#endif

/*
 * File Name: Employee.h
 * Assignment: Lab 1 Exercise C
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <string>
using namespace std;

enum EmployeeState {
    ACTIVE, SUSPENDED, RETIRED, FIRED
};

class Employee {
    private:
        string name;
        string address;
        EmployeeState state;
        friend class Company;
};

#endif
```

```
/*
 * File Name: Customer.h
 * Assignment: Lab 1 Exercise C
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#ifndef CUSTOMER_H
#define CUSTOMER_H
#include <string>
using namespace std;
class Customer{
    private:
        string name;
        string  phone;
        string address;
        friend class Company;
        friend class Employee;
};

#endif
```

Exercise D

Code

```
/*
 * File Name: Point.h
 * Assignment: Lab 1 Exercise D
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#ifndef POINT_H
#define POINT_H
#include <iostream>
class Point
{
    private:
        double x; // x coordinate of a location on Cartisian Plain
        double y; // y coordinate of a location on Cartisian Plain
        friend class Human;
        //setters
        // private so human can access these setters, but they're hidden
        void set_y(const double);
        void set_x(const double);
    public:
        Point();
        Point(const double, const double);
```

```
        //getters
        double get_x() const;
        double get_y() const;
};

#endif

/*
 * File Name: Point.cpp
 * Assignment: Lab 1 Exercise D
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#include "Point.h"
//constructors
Point::Point() : x(0), y(0) {}
Point::Point(const double a, const double b) : x(a), y(b) {}
//getters
double Point::get_x() const {
    return x;
}
double Point::get_y() const {
    return y;
}
//setters
void Point::set_x(const double a) {
    x = a;
}
void Point::set_y(const double a) {
    y = a;
}


/*
 * File Name: Human.h
 * Assignment: Lab 1 Exercise D
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#ifndef HUMAN_H
#define HUMAN_H
#include <cstring>
#include "Point.h"
class Human {
    protected:
        Point location; // Location of an object of Human on a Cartisian
Plain
        char *name;     // Human's name
    public:
        //constructors
        Human();
        Human(const char*);
```

```
        Human(const char *nam = "", const double x = 0, const double y =
0);
        ~Human(); //destructor

        //getters
        const char *get_name() const;
        // TODO: const return?
        Point get_point() const;

        //setters
        void set_x(const double);
        void set_y(const double y);
        void set_name(const char *);

        virtual void display();
};

#endif


/*
 * File Name: Human.cpp
 * Assignment: Lab 1 Exercise D
 * Completed By: Beau McCartney, Apostolos Scondriannis
 * Submission Date: September 24th, 2021
 */

#include <cstring>
#include <iostream>
#include "Human.h"

using namespace std;

Human::Human() {
    location.set_x(0);
    location.set_y(0);
    name = new char[1];
    strcpy(this->name, "");
}
Human::Human(const char * nam) {
    location.set_x(0);
    location.set_y(0);
    name = new char[strlen(nam) + 1];
    strcpy(this->name, nam);
}

Human::Human(const char *nam, const double a, const double b) {
    location.set_x(a);
    location.set_y(b);
    name = new char[strlen(nam) + 1];
    strcpy(this->name, nam);
}
Human::~Human() {
    delete []name;
```

```
}

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

void Human::set_name(const char *name) {
    location.set_x(0);
    location.set_y(0);
    delete []name;
    this->name = new char[strlen(name) + 1];
    strcpy(this->name, name);
}

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

void Human::set_x(const double x) {
    location.set_x(x);
}

void Human::set_y(const double y) {
    location.set_y(y);
}

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