Course: Principles of Software Design – ENSF 480

Lab 1

Instructor Name: M. Moussavi

Student Name: Gerardo Garcia de Leon

Lab Section B02

Date Submitted: September 13th 2024

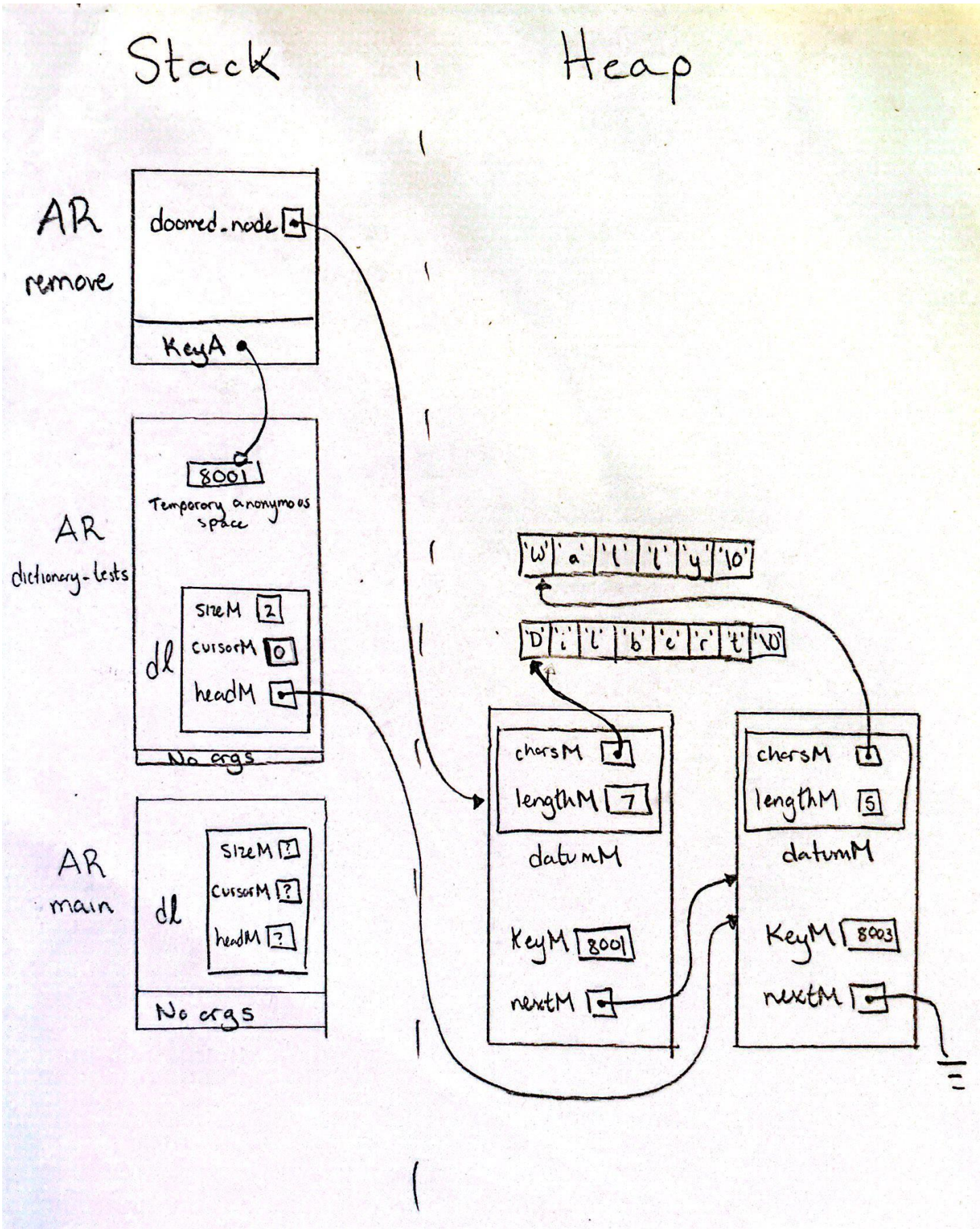
EXERCISE A:

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.**	Line 18. The line Mystring x[2] calls the default constructor Mystring::Mystring();
constructor with char* argument is called.	Line 22. Mystring* z = new Mystring("4") causes a call to the constructor Mystring::Mystring(const char* s)
copy constructor is called. **copy constructor is called.**	Line 24. By using the append function, a copy is created of both the passing arguments (*z) and (x[1])
destructor is called. **destructor is called.**	Line 25. After the copies of the previous objects go out of scope, the destructor is called
copy constructor is called.	Line 26. The initialization of the object mars is a deep copy of the value held in x[0]
assignment operator called.	Line 28. The Mystring& Mystring::operator = (const Mystring& s) constructor is used as the object already exists, so we use the assignment operator
constructor with char* argument is called. **constructor with char* argument is called.**	Line 30 and 32. Mystring jupiter("White") and ar[0] = new Mystring("Yellow") both use the Mystring::Mystring (const char* s) constructor
destructor is called. **destructor is called.** **destructor is called.** **destructor is called.** **destructor is called.**	Line 34. This bracket is the end of the code block, causing the objects inside to go out of scope and calling the destructor in the process (x[0], x[1], z, mars, jupiter)
constructor with char* argument is called.	Line 39. Mystring d = "Green" initializes d using the Mystring::Mystring (const char* s) constructor
Program terminated successfully.	Line 41. The cout << "\nProgram terminated successfully." <<endl prints the line as stated

destructor is called. **destructor is called**	Line 43. Once the program ends, the remaining objects go out of scope and call the destructor

EXERCISE B:

PART 1 :


Stack
Heap
AR
remove
doomed_node
KeyA
8001
Temporary anonymous space
AR
dictionary-tests
dl
sizeM 2
cursorM 0
headM
No args
AR
main
dl
sizeM ?
cursorM ?
headM ?
No args
'W' 'a' 'l' 'l' 'y' '\0'
'D' 'i' 'l' 'b' 'e' 'r' 't' '\0'
charsM
lengthM 7
datumM
KeyM 8001
nextM
charsM
lengthM 5
datumM
KeyM 8003
nextM

PART 2 :

I will only add the parts that were edited as it is a huge amount of space to include the complete files

```
/* File name: dictionaryList.cpp
  Assignment: Lab 1 Exercise B
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/
void DictionaryList::find(const Key& keyA)
{
	if(headM == 0){
		cout << "There is no list to search" << endl;
		exit(1);
	}

	for(Node* ptr = headM; ptr != 0; ptr = ptr ->nextM){
		if(keyA == ptr -> keyM){ //If keyA matches the value held by the current nodes's keyM, set cursorM to that node
			cursorM = ptr;
			return;
		}
	}
	cursorM = NULL; //We didn't find a key, so set to null
}


void DictionaryList::destroy()
```

```
{
  while(headM != 0){
        Node* temp = headM;
        headM = headM -> nextM;
        delete temp;
  }

  headM = 0; //headM should already be 0 after the while loop, but just to be safe we set it to zero
}



void DictionaryList::copy(const DictionaryList& source)
{
  if(source.headM == 0){
        this -> headM = 0;
        this -> cursorM = 0;
        this -> sizeM = 0;
        return;
  }

  this -> headM = new Node (source.headM -> keyM, source.headM -> datumM, nullptr); //Creating the first node
  Node* next_src_node = source.headM -> nextM;
  Node* next_cpy_node = this -> headM;

  while(next_src_node != 0){ //As long as there's nodes to copy
        next_cpy_node -> nextM = new Node(next_src_node -> keyM, next_src_node -> datumM, nullptr);
        next_cpy_node = next_cpy_node -> nextM;
```

```
        next_src_node = next_src_node -> nextM;
 }

 this -> sizeM = source.sizeM; //Copying the size

 if(source.cursorM == 0){ // If there is no cursor set, we set our copy's cursor to 0
        this -> cursorM = 0;
 }
 else{ // We use two iterators going at the same speed to stop when we find the source cursor. I'm sure there is a more efficient way of doing this but I can't think of it
        Node* src_cursor = source.headM;
        Node* cpy_cursor = this -> headM;

        while(src_cursor != source.cursorM){
                src_cursor = src_cursor -> nextM;
                cpy_cursor = cpy_cursor -> nextM;
        }
        this -> cursorM = cpy_cursor;
 }
}
```

OUTPUT:

```
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:

For all of the following source code files, in the “public” section of variables, I only wrote “...” to show that there would be member functions in that section, but in the exercise it says not to worry about implementation.

```
/* File name: company.h
  Assignment: Lab 1 Exercise C
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/

//company.h
#ifndef COMPANY_H
#define COMPANY_H
```

```
#include <string>
#include <vector>
#include "address.h"
#include "date.h"
#include "customer.h"
#include "employee.h"
using namespace std;

class Company{
        private:
        string name;
        Address address;
        Date dateEstablished;
        vector <Customer> customers;
        vector <Employee> employees;

        public:
        ...
}

#endif

/* File name: address.h
  Assignment: Lab 1 Exercise C
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/
```

```
//address.h
#ifndef ADDRESS_H
#define ADDRESS_H

#include <string>
using namespace std;

class Address {
private:
   string street;
   string city;
   string province;
   string postalCode;
   string country;
public:
   ...
};

#endif

/* File name: date.h
  Assignment: Lab 1 Exercise C
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/

//date.h
```

```
#ifndef DATE_H
#define DATE_H

#include <string>
using namespace std;

class Date {
private:
   string day;
   string month;
   string year;
public:
   ...
};

#endif

/* File name: employee.h
  Assignment: Lab 1 Exercise C
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/

//employee.h
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
```

```
#include "address.h"
#include "date.h"
using namespace std;

class Employee {
private:
   string name;
   Address address;
   string state;
   Date birthdate;
public:
   ...
};

#endif

/* File name: customer.h
  Assignment: Lab 1 Exercise C
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 11th, 2024
*/

//customer.h
#ifndef CUSTOMER_H
#define CUSTOMER_H

#include <string>
#include "address.h"
```

```
using namespace std;

class Customer {
private:
   string name;
   string phone;
   Address address;
public:
   ...
};

#endif
```

EXERCISE D:

```
/* File name: fixed_point.h
  Assignment: Lab 1 Exercise D
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 12th, 2024
*/

#ifndef POINT_H
#define POINT_H

#include <cstring>
using namespace std;

class Point{
```

```
        friend class Human; // This declaration allows the class Human to access the private x and y variables

private:
  double x;
  double y;
public:
  Point(double a, double b);
  double get_x() const; // Added const to clarify the function does not manipulate member variables
  double get_y() const; // Added const to clarify the function does not manipulate member variables
  void set_x(double a);
  void set_y(double y);
};

#endif

/* File name: fixed_point.cpp
   Assignment: Lab 1 Exercise D
   Lab Section: B02
   Completed by: Gerardo Garcia de Leon
   Development Date: Sep 12th, 2024
*/

#include "fixed_point.h"
using namespace std;

Point::Point(double a = 0, double b = 0): 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: fixed_human.h
  Assignment: Lab 1 Exercise D
  Lab Section: B02
  Completed by: Gerardo Garcia de Leon
  Development Date: Sep 12th, 2024
*/

#ifndef HUMAN_H
#define HUMAN_H

#include <cstring>
#include <iostream>
using namespace std;

class Human{
protected:
 Point location;
 char* name;

public:
 Human(); //Added a default constructor to have default values
 Human(const char* name, double x, double y);
 ~Human(); // Added a destructor
 Human& operator =(const Human& other); // Added an assignment operator
 Human(const Human& other); // Added copy constructor
 const char* get_name() const; // Added const keyword to encapsulate
```

```
  void set_name(const char* name); // Added const keyword to ensure no accidental changes to the name argument
  const Point get_point() const; // Added const to encapsulate
  void display() const; // Added const as it does not change member values
}

#endif

/* File name: fixed_human.cpp
   Assignment: Lab 1 Exercise D
   Lab Section: B02
   Completed by: Gerardo Garcia de Leon
   Development Date: Sep 12th, 2024
*/

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

Human::Human() : name(new char[1]) {
    name[0] = '\0'; // Empty string just to allocate some memory
    location.set_x(0);
    location.set_y(0);
}

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

```
    location.set_y(y);
}

Human::~Human() {
    delete[] name; // Deletes the dynamically allocated data to prevent memory leak
}

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

Human& Human::operator=(const Human& other) {
    if (this != &other) {
        delete[] this -> name; // Delete existing memory
        this -> name = new char[strlen(other.name) + 1];
        strcpy(this -> name, other.name);
        location.set_x(other.location.get_x());
                    location.set_y(other.location.get_y());
    }
    return *this;
}

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

void Human::set_name(const char* name) {
```

```
    delete[] this -> name; // Delete existing memory
    this -> name = new char[strlen(name) + 1];
    strcpy(this -> name, name);
}

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

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

/* File name: fixed_main.cpp
   Assignment: Lab 1 Exercise D
   Lab Section: B02
   Completed by: Gerardo Garcia de Leon
   Development Date: Sep 12th, 2024
*/
#include <iostream>
#include "fixed_human.h"
#include "fixed_point.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;
}
```