Names: Neha Parmar, Shahed Issa

Course Name: Principles of Software Design

Lab Section: B01

Course Code: ENSF 480

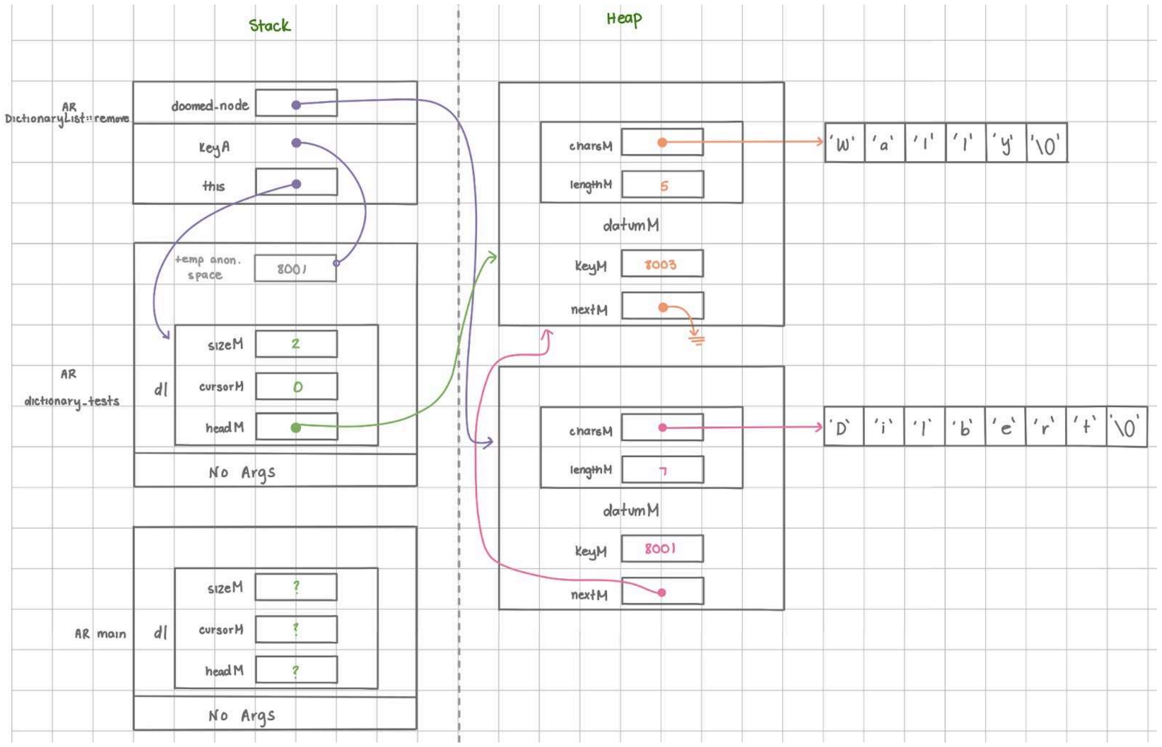
Assignment Number: Lab-1

Submission Date and Time: 13/09/2024

Exercise A:

Program output and its order	Your explanation (why and where is the cause for this output)
constructor with int argument is called.	Line 12 in exAmain: Mystring c = 3; This calls the constructor Mystring::Mystring(int n), which makes an empty string with a total capacity of 3.
default constructor is called. default constructor is called.	Line 18 in exAmain: Mystring x[2]; This calls the default constructor: Mystring::Mystring() twice to create an array of two MyString objects.
constructor with char* argument is called.	Line 22 in exAmain: Mystring* z = new Mystring("4"); This calls the constructor Mystring::Mystring(const char *s), which creates a new MyString object initialized with the string "4"
copy constructor is called. copy constructor is called.	Line 24 in exAmain: x[0].append(*z).append(x[1]); The append function creates a temp variable that requires the copy constructor in Line 98 in mystring.cpp: char *tmp = new char [lengthM + other.lengthM + 1]; So to append x[0] and x[1], there needs to be two calls to the copy constructor.
destructor is called. destructor is called.	After the append function in mystring.cpp. The MyString destructor deallocates memory for charsM when it is done copying, so once for x[0] and once for x[1].
copy constructor is called.	Line 26 in exAmain: Mystring mars = x[0]; This creates a copy of x[0] into the mars object.
assignment operator called.	Line 28 in exAmain: x[1] = x[0]; This is the assignment operator that copies the content of x[0] to x[1].
constructor with char* argument is called. constructor with char* argument is called.	Line 30 in exAmain: Mystring jupiter("White"); This calls the constructor Mystring::Mystring(const char *s) which creates an object "jupiter" with the string "White". Line 32 in exAmain: ar[0] = new Mystring ("Yellow"); This calls the constructor Mystring::Mystring(const char *s) which creates an array with the string "Yellow".
destructor is called. destructor is called. destructor is called. destructor is called. destructor is called.	Line 35: x[2], mars, and jupiter go out of scope and Line 37: ar is deleted so the MyString destructor is called and they are deleted so the memory is freed.
constructor with char* argument is called.	Line 39 in exAmain: Mystring d = "Green"; This calls the constructor Mystring::Mystring(const char *s) which creates an object "d" with the string "Green".
Program terminated successfully.	Line 41: cout << "\nProgram terminated successfully." <<endl;
destructor is called. destructor is called	Line 43: Destructors for c and d are called.

Exercise B:
Part I:



Part II:
Source Code:

```
void DictionaryList::find(const Key &keyA)
{
  if (headM == 0)
  {
    cursorM = 0;
    return;
  }

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

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

    while (next != 0 && keyA >= prev->keyM)
    {

      if (prev->keyM == keyA)
      {
        cursorM = prev;
        return;
      }

      prev = next;
      next = prev->nextM;
    }

    if (prev->keyM == keyA)
    {
      cursorM = prev;
      return;
    }
  }

  cursorM = 0;
  return;
}

void DictionaryList::destroy()
{
  if (headM == 0)
  {
    return;
  }

  cursorM = 0;
  sizeM = 0;

  Node *prev = headM;
  Node *next = prev->nextM;

  do
  {

    delete prev;

    prev = next;
    next = prev->nextM;
  } while (next != 0);

  delete prev;
  headM = 0;
}

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

  if (source.headM == 0)
  {
    this->headM = 0;
    this->cursorM = 0;
    this->sizeM = 0;
    return;
  }

  this->cursorM = 0;
  this->sizeM = 0;
  Node *prev = source.headM;
```

```
    Node *next = prev->nextM;

    Node *copy = new Node(prev->keyM, prev->datumM, 0);
    Node *head = copy;
    this->sizeM++;

    while (next != 0)
    {

      copy->nextM = new Node(next->keyM, next->datumM, 0);

      if (source.cursorM == prev)
      {
        this->cursorM = copy;
      }

      copy = copy->nextM;
      this->sizeM++;

      prev = next;
      next = prev->nextM;
    }

    copy->nextM = next;
    this->headM = head;
  }
```

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:

```
#include <string>
#include <vector>
using namespace std;

class Name {
  private:
    string firstName;
    string lastName;.
};

class Address {
  private:
    string street;
    string city;
    string state;
    string zipCode;
};

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

class Person {
  private:
    Name name;
    Address address;
};

class Employee : public Person {
  private:
    Date dateOfBirth;
    string currentState;
};

class Customer : public Person {
  private:
    string phone;
};

class Company {
  private:
    string companyName;
    Address companyAddress;
    Date dateEstablished;
    vector<Employee> employees;
    vector<Customer> customers; };
```

Exercise D:

Point.cpp:

```
/*
 *File Name: Point.cpp
 * Assignment: Lab 1 Exercise D
 * Completed by: Shahed Issa and Neha Parmar
 * Submission Date: Sept 11, 2024
 */

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

class Point{

    Point::Point() : x(0), y(0) {}

    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;
    }
};
```

Point.h:

```
/*
 *File Name: Point.h
 * Assignment: Lab 1 Exercise D
 * Completed by: Shahed Issa and Neha Parmar
 * Submission Date: Sept 11, 2024
 */

#ifndef POINT_H
#define POINT_H

class Point
{
private:
   friend class Human;
   double x; // x coordinate of a location on Cartisian Plain
   double y; // y coordinate of a location on Cartisian Plain

   Point();                   // PROMISES: assigns the values
x=0and y=0 as default values.
   Point(double a, double b); // PROMISES: assigns the values of
x=a and y=b.
   void set_x(double a);      // PROMISES: Sets the value of x =
a.
   void set_y(double a);      // PROMISES: Sets the value of y =
a.

public:
   double get_x() const;
   // PROMISES: Returns the value of the x coordinate.

   double get_y() const;
   // PROMISES: Returns the value of the y coordinate.
};

#endif // POINT_H
```

Human.cpp:

```
/*
 *File Name: Human.cpp
 * Assignment: Lab 1 Exercise D
 * Completed by: Shahed Issa and Neha Parmar
 * Submission Date: Sept 11, 2024
 */

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

Human::Human() {
    strcpy(this->name, "");
    location.set_x(0);
    location.set_y(0);
}

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

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

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

```
void Human::set_name(const char *name) {
    // Delete the old memory
     delete[] this->name;

    // Allocate new memory
    this->name = new char[strlen(new_name) + 1];

    // Copy the new name
    strcpy(this->name, new_name);
}

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

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

Human.h:

```
/*
 *File Name: Human.h
 * Assignment: Lab 1 Exercise D
 * Completed by: Shahed Issa and Neha Parmar
 * Submission Date: Sept 11, 2024
 */

#ifndef HUMAN_H
#define HUMAN_H
#include "Point.h"

class Human {
private:
    Point location;     // Location of the Human on a Cartesian
plane
    char *name;         // Name of Human

public:
    Human(); // PROMISES: Initializes the name with null and sets
the location with coordinates (0,0).

    Human(const char *nam, double x, double y);  // PROMISES:
Initializes the name with the provided string and sets the
location with the provided coordinates.

    ~Human(); // PROMISES: Releases dynamically allocated memory


    char *get_name();  // PROMISES: Returns a pointer to the name
of the Human.
```

```
    void set_name(const char *name);  // PROMISES: Deletes any
existing name memory and allocates new memory, then stores the new
name there.

    Point get_point() const; // PROMISES: Returns a copy of the
Point object (location) of the Human.

    void display(); // PROMISES: Outputs the name and location of
the Human to the standard output.
};

#endif // HUMAN_H
```

main.cpp:

```
#include <iostream>
#include <cstring>
#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;
}
```