CSE214

HOMEWORK 7 - SPRING 2020

HOMEWORK 7 - due Friday, May 8th no later than 6:00PM

REMINDERS:

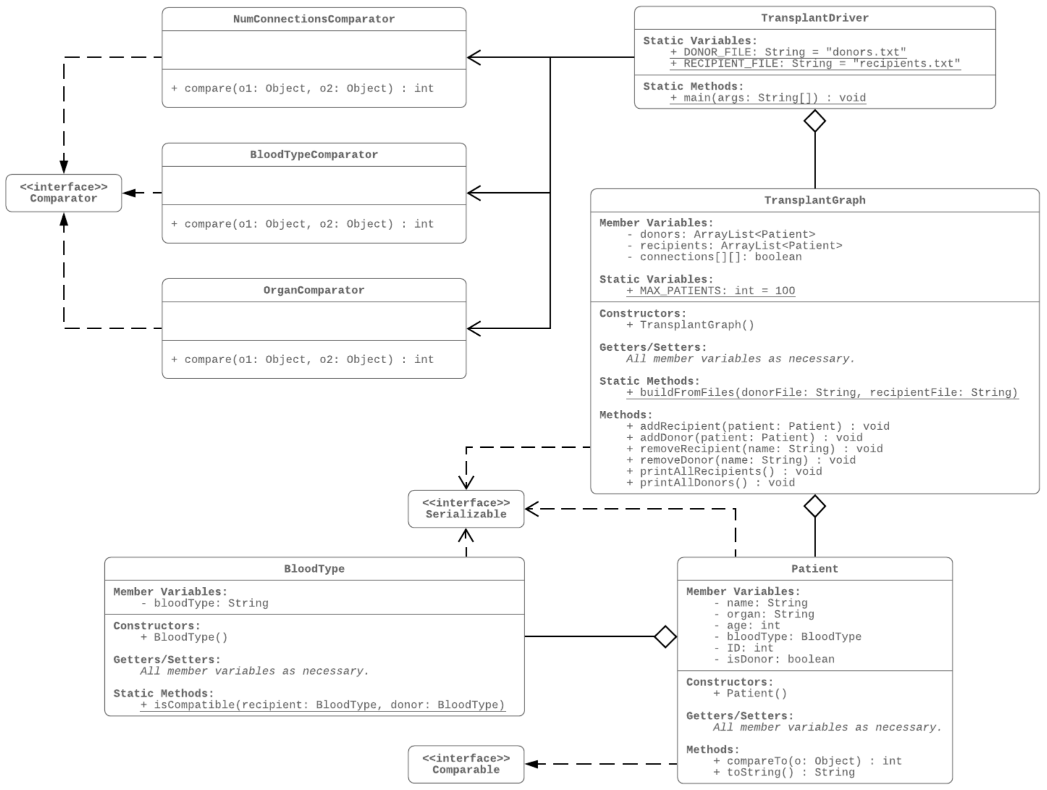
- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may use any Java API class that you wish.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

You have been tasked with the creation of a system which will be responsible for matching organ donors with potential transplant recipients. In order to achieve high success rates of surgeries, there are certain parameters which must be considered before matching patients. This includes making sure that a patient can receive a transplant from an organ donor based on both the patient's and donor's blood types. If these blood types are not compatible, the patient cannot receive the transplant, and no match between the two is made. To better understand how blood type compatibility works, a table is included (see below) to assist you in determining compatibility.

You will store the information about donors and recipients as a graph. Each node will represent either a donor or a recipient, with the neighbors of that node being those who have been determined to be a match for the needed transplant. You should also be able to display the information to the user, in a variety of sorted formats (by blood type, by ID, by number of matches, by organ).

UML

The UML Diagram for all the classes as specified below is as follows:


NumConnectionsComparator
+ compare(o1: Object, o2: Object) : int
BloodTypeComparator
+ compare(o1: Object, o2: Object) : int
OrganComparator
+ compare(o1: Object, o2: Object) : int
<<interface>> Comparator
TransplantDriver
Static Variables:
+ DONOR_FILE: String = "donors.txt"
+ RECIPIENT_FILE: String = "recipients.txt"
Static Methods:
+ main(args: String[]) : void
TransplantGraph
Member Variables:
- donors: ArrayList<Patient>
- recipients: ArrayList<Patient>
- connections[][]: boolean
Static Variables:
+ MAX_PATIENTS: int = 100
Constructors:
+ TransplantGraph()
Getters/Setters:
All member variables as necessary.
Static Methods:
+ buildFromFiles(donorFile: String, recipientFile: String)
Methods:
+ addRecipient(patient: Patient) : void
+ addDonor(patient: Patient) : void
+ removeRecipient(name: String) : void
+ removeDonor(name: String) : void
+ printAllRecipients() : void
+ printAllDonors() : void
<<interface>> Serializable
BloodType
Member Variables:
- bloodType: String
Constructors:
+ BloodType()
Getters/Setters:
All member variables as necessary.
Static Methods:
+ isCompatible(recipient: BloodType, donor: BloodType)
Patient
Member Variables:
- name: String
- organ: String
- age: int
- bloodType: BloodType
- ID: int
- isDonor: boolean
Constructors:
+ Patient()
Getters/Setters:
All member variables as necessary.
Methods:
+ compareTo(o: Object) : int
+ toString() : String
<<interface>> Comparable

SPECIFICATIONS

1. public class Patient

Write a fully-documented class `Patient` which represents an active organ donor or recipient in the database. This class should `implement Comparable` and provide a `compareTo` method, so that it may be easily sorted by the `ID` field. The Patient object should contain the following data fields:

- `private String name` - **The name of the donor or recipient**
- `private String organ` - **The organ the patient is donating or receiving**
- `private int age` - **The age of the patient**
- `private BloodType bloodType` - **The blood type of the patient**
- `private int ID` - **The ID number of the patient. If a donor, this must be unique amongst all donors. If a recipient, this must be unique amongst all recipients. This is used to map this patient to an index in the adjacency matrix, as it denotes which row or column this patient's connections are stored in.**
- `private boolean isDonor` - `true` **if this Patient is a donor,** `false` **if a recipient**

In addition, you should implement the following methods:

- `public Patient()` - **Constructor. You may include additional constructors that take parameters as necessary.**
- **Getters and setters, as necessary.**
- `public int compareTo(Object o)` - **Compares `this` Patient object to `o`, comparing by `ID` such that the values should be sorted in ascending order.**
- `public String toString()` - **Returns a String representation of this Patient object. See Sample IO for details.**

2. public class BloodType

Write a fully-documented class `BloodType`. This class will contain a `String` member variable `bloodType` to denote a specific blood type, and a static method which can be invoked to determine if two blood types are compatible with each other.

- `public BloodType()` - **Constructor. You may include additional constructors that take parameters as necessary.**
- `public static boolean isCompatible(BloodType recipient, BloodType donor)`
  - **Determines whether two blood types are compatible, returning true if they are, and false otherwise.**
  - **The following table can be used to determine compatability:**

	Donor O	Donor A	Donor B	Donor AB
Recipient O	TRUE	FALSE	FALSE	FALSE
Recipient A	TRUE	TRUE	FALSE	FALSE
Recipient B	TRUE	FALSE	TRUE	FALSE
Recipient AB	TRUE	TRUE	TRUE	TRUE

3. public class TransplantGraph

Write a fully-documented class `TransplantGraph` that contains an adjacency matrix for the organ donors and recipients. Include the following variables:

- `private ArrayList<Patient> donors` - **Contains all organ donors in our system**
- `private ArrayList<Patient> recipients` - **Contains all recipients in our system**
- `public static final int MAX_PATIENTS = 100` - **The maximum number of donors or recipients our system can store**
- `private boolean[][] connections` - **Adjacency matrix used to track compatibility. You should set up your matrix such that the rows are used to store information about the donors, and the columns are used to store information about the recipients (ie, if you read row 0 left to right, each cell will denote whether donor 0 is compatible with a recipient. Likewise, if you read column 0 from top to bottom, each cell will denote whether recipient 0 is compatible with a donor). Doing so, `connections[i][j]` represents whether or not a link between donor i and recipient j exists.**
- **NOTE: The connections matrix is used to map connections between donors and recipients, while the donors and recipients lists are used to actually store the information associated with those patients.**

You should implement the following methods:

- `public TransplantGraph()` - **Constructor. You may include additonal constructors that take parameters as necessary.**
- `public static TransplantGraph buildFromFiles(String donorFile, String recipientFile)`
  - **Creates and returns a new `TransplantGraph` object, intialized with the donor information found in `donorFile` and the recipient information found in `recipientFile`.**
- `public void addRecipient(Patient patient)` - **Adds the specified Patient to the recipients list. This method must also update the connections adjacency matrix, as necessary.**

- `public void addDonor(Patient patient)` - Adds the specified Patient to the donors list. This method must also update the connections adjacency matrix, as necessary.
- `public void removeRecipient(String name)` - Removes the specified Patient from the recipients list. This method must also update the connections adjacency matrix, removing all connections to this recipient, and removing the column associated with the patient from the matrix.
- `public void removeDonor(String name)` - Removes the specified Patient from the donors list. This method must also update the connections adjacency matrix, removing all connections to this donor, and removing the row associated with the patient from the matrix.
- `public void printAllRecipients()` - Prints all organ recipients' information in a neatly formatted table. See Sample I/O.
- `public void printAllDonors()` - Prints all organ donors' information in a neatly formatted table. See Sample I/O.

NOTE: When adding an entry to the adjacency matrix, the item should be placed at the next available slot, with no gaps between elements. If you already have 4 donors (located at row indices 0-3), the next donor should be stored stored at index 4.
NOTE: When removing an entry from the adjacency matrix, all other rows and columns should be shifted so that there are no gaps between elements. If you have 4 recipients (located at column indices 0-3) and remove the recipient at column index 1, then column 2 should be moved to column 1, and column 3 should be moved to column 2. The `ID` field of each Patient object moved this way will also need to be updated, so that the new `ID` field now refers to the new location in the matrix.

4. Comparators

Write fully-documented Comparator classes that will be used to sort your patient lists. See the "Sample Comparable/Comparator Code" section below.

- `public class NumConnectionsComparator implements Comparator<Patient>`
- `public class BloodTypeComparator implements Comparator<Patient>`
- `public class OrganComparator implements Comparator<Patient>`

NOTE: After reordering your lists of donors or recipients, the indices of items in the list may no longer correctly map to their `ID` values, also causing the adjacency matrix entries to refer to incorrect patients. As such, you will need to either update all connection information in the adjacency matrix or revert to the lists' initial states before performing any other operations on the matrix.

5. public class TransplantDriver

Write a fully-documented class `TransplantDriver` which will act as the main driver for the application. This class contains the main method for the program, which will first attempt to load any previously serialized TransplantGraph object located in 'transplant.obj'. If such an object is not found, you should instead construct a new `TransplantGraph` object using the information stored in the files denoted by `DONOR_FILE` and `RECIPIENT_FILE` by calling the static method `buildFromFiles` in the TransplantGraph class. The program should serialize its TransplantGraph object on program termination, so that it may be loaded at a later time. You should declare the follow variables:

- `public static final String DONOR_FILE = "donors.txt"`
- `public static final String RECIPIENT_FILE = "recipients.txt"`

This class should contain the `main` method, and act as a menu-driven application, prompting the user for input among the following options:

- (LR) - List all recipients: Displays a table of all patients in the recipients list.
- (LO) - List all donors: Displays a table of all patients in the donors list.
- (AO) - Add new donor: Adds a new donor to the system.
- (AR) - Add new recipient: Adds a new recipient to the system.
- (RO) - Remove donor: Removes a donor from the system.
- (RR) - Remove recipient: Removes a recipient from the system.
- (SR) - Sort recipients: Displays a submenu:
  - (I) Sort by ID
  - (N) Sort by Number of Donors
  - (B) Sort by Blood Type
  - (O) Sort by Organ Needed
  - (Q) Back to Main Menu
- (SO) - Sort donors: Displays a submenu:
  - (I) Sort by ID
  - (N) Sort by Number of Recipients
  - (B) Sort by Blood Type
  - (O) Sort by Organ Donated
  - (Q) Back to Main Menu
- (Q) - Quit

6. You will need classes to handle the exceptions thrown (see class specifications above for exception classes you need).

NOTE: You may include additional methods, variables, or classes as necessary or as you find convenient.

SAMPLE COMPARABLE/COMPARATOR CODE

```
/**
 * Here we see how we can use Comparable and Comparators to sort in
 * different ways. We can use the sorting method supplied by the
 * Employee class, or we can call Collections.sort with a
 * specific Comparator to change the sorting behavior.
 **/
import java.util.*;

public class CollectionsTester {
    public static void main(String[] args) {
        ArrayList staff = new ArrayList();

        staff.add(new Employee("Joe", 100000, 177700010));
        staff.add(new Employee("Jane", 200000, 111100010));
        staff.add(new Employee("Bob", 66666, 1999000010));
        staff.add(new Employee("Andy", 77777, 188800010));

        Collections.sort(staff);                                    // Sort by salary
        System.out.println("Lowest paid employee: "+staff.get(0));    // Prints Bob

        Collections.sort(staff, new NameComparator());               // Sort by aplahabetical order
        System.out.println("First employee in list: "+staff.get(0));  // Prints Andy

        Collections.sort(staff, new IdComparator());                 // Sort by ID number
        System.out.println("Employee with lowest ID: "+staff.get(0)); // Prints Jane
    }
}

public class Employee implements Comparable {
    private String name;
    private int salary;
    private int id;
    public Employee(String initName, int initSal, int initId) {
        id  = initId;
        name = initName;
        salary = initSal;
    }
    public String getName(){ return name; }
    public int getSalary() { return salary; }
    public int getId(){ return id; }
    public void setSalary(int newSalary) {
        salary = newSalary;
    }
    public int compareTo(Object o) {
        // The compareTo method supplied by the Employee class sorts by salary
        Employee otherEmp = (Employee)o;
        if (this.salary == otherEmp.salary)
            return 0;
        else if (this.salary > otherEmp.salary)
            return 1;
        else
            return -1;
    }
    public String toString() {
        return name + ", $" + salary + ", "+ id;
    }
}

public class NameComparator implements Comparator {
    // The compare method supplied here sorts by name
    public int compare(Object o1, Object o2) {
        Employee e1 = (Employee) o1;
        Employee e2 = (Employee) o2;
        return (e1.getName().compareTo(e2.getName()));
    }
}

public class IdComparator implements Comparator {
    // The compare method supplied here sorts by ID
    public int compare(Object o1, Object o2) {
        Employee e1 = (Employee) o1;
        Employee e2 = (Employee) o2;
        if (e1.getId() == e2.getId())
            return 0;
        else if (e1.getId() > e2.getId())
            return 1;
        else
            return -1;
        }
    }
```

ABOUT SERIALIZATION

Serialization is way of converting a state representation of an object into a byte stream. Deserialization is the reverse process, where a byte stream is used to recreate the original Java object in memory. Since we want to be able to save all of our patient donor and recipient records when quitting the application, you will be implementing Serializable to enable serialization and deserialization of your classes. You should write your TransplantGraph object to a "transplant.obj" file when quitting the program, and attempt to load a TransplantGraph object upon starting the program if a "transplant.obj" file is found.

In order to support serialization, any class that you wish to serialize must implement Serializable. You do not have to write any custom methods while implementing this interface, as Java does provide a default implementation. Although it is not required, you may find it helpful to declare a `static final long serialVersionUID`. This variable is used to ensure that the current version of the class is compatible with the serialized version of the object. Certain changes to the class structure (such as removing, renaming, or adding member variables) may make it so that the readObject() method is unable to correctly process an old version. Note that you can omit this variable if you choose and Java will calculate a default value for it.

A generalized structure for a class which implements the Serializable interface would be as follows:

```
import java.io.Serializable;

public class MySerializableClass implements Serializable
{
    public static final long serialVersionUID = 1L; // Version 1
    // Class variables/methods/etc as usual
}
```

To serialize data:

```
FileOutputStream file = new FileOutputStream("data.obj");
ObjectOutputStream fout = new ObjectOutputStream(file);

fout.writeObject(objToWrite); // Here "objToWrite" is the object to serialize
fout.close();
// Note that if data.obj does not exist, this will automatically create it and write into it.
```

To deserialize data:

```
try {
    FileInputStream file = new FileInputStream("data.obj");
    ObjectInputStream fin  = new ObjectInputStream(file);
    obj = (MyObjectType) fin.readObject();
    fin.close();
} catch(IOException e) {
    // This exception is thrown if data.obj does not exist.
    // Since there is nothing to load, here you should just create a new object, instead.
}
```

TEXT FILE FORMAT

You can download a sample dataset here. The text files will contain lines of the following format:

```
ID, NAME, AGE, ORGAN, BLOODTYPE
```

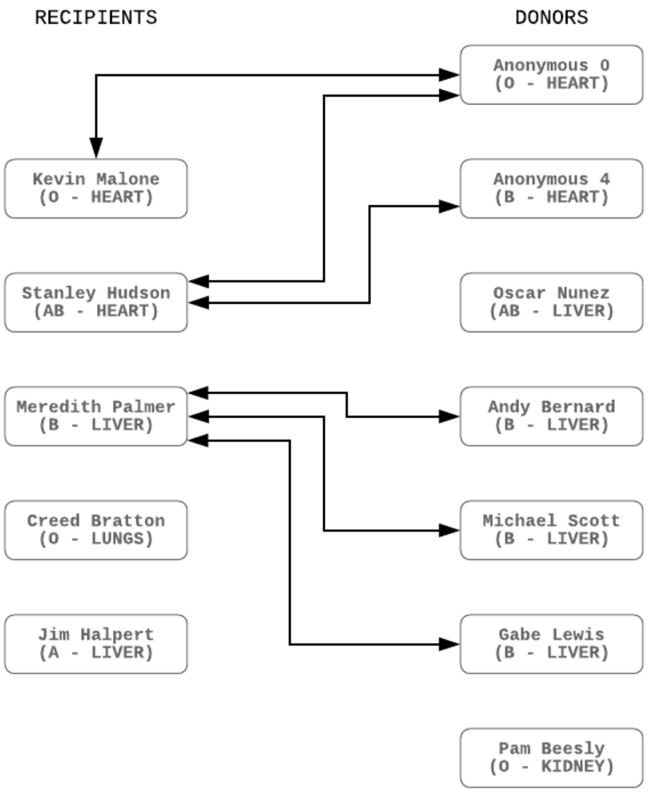
The initial recipients.txt file will contain:

```
0, Kevin Malone, 45, Heart, O
1, Stanley Hudson, 60, Heart, AB
2, Meredith Palmer, 38, Liver, B
3, Creed Bratton, 70, Lungs, O
4, Jim Halpert, 34, Liver, A
```

The initial donors.txt file will contain:

```
0, Anonymous, 43, Heart, O
1, Pam Beesly, 28, Kidney, O
2, Andy Bernard, 38, Liver, B
3, Oscar Nunez, 33, Liver, AB
4, Anonymous, 21, heart, B
5, Michael Scott, 47, liver, B
6, Gabe Lewis, 26, liver, B
```

When constructing the graph from these initial files, your graph should have the following structure:



You may use the following code to open an input stream for reading from a text file:

```
FileInputStream fis = new FileInputStream(filename);
InputStreamReader instream = new InputStreamReader(fis);
BufferedReader reader = new BufferedReader(instream);
```

Once the stream is open, you an read one line at a time as follows:

```
String data = reader.readLine();
```

You can also create the reader using a scanner:

```
Scanner scanner = new Scanner (new File(filename));
```

You can then use the Java String class to tokenize the input as necessary.

INPUT FORMAT

- **Each menu operation is entered on its own line and should be case insensitive (i.e. 'q' and 'Q' are the same).**
- **All text input should be treated as case insensitive (name, organ, blood type, etc).**

OUTPUT FORMAT

- **Each command should output the result (as shown in the sample IO below) after each operation is performed.**
- **All menu operations must be accompanied by a message indicating what operation was performed and whether or not it was successful.**

SAMPLE INPUT/OUTPUT

```
// Comments in green, input in red,, output in black.

transplant.obj not found. Creating new TransplantGraph object...
Loading data from 'donors.txt'...
Loading data from 'recipients.txt'...

Menu:
    (LR) - List all recipients
    (LO) - List all donors
    (AO) - Add new donor
    (AR) - Add new recipient
    (RO) - Remove donor
    (RR) - Remove recipient
    (SR) - Sort recipients
    (SO) - Sort donors
    (Q) - Quit

Please select an option: LR

Index | Recipient Name     | Age | Organ Needed  | Blood Type | Donor ID
========================================================================
   0  | Kevin Malone       | 45  |  Heart        |     O      | 0
   1  | Stanley Hudson     | 40  |  Heart        |     AB     | 0, 4
   2  | Meredith Palmer    | 38  |  Liver        |     B      | 2, 5, 6
   3  | Creed Bratton      | 70  |  Lungs        |     O      |
   4  | Jim Halpert        | 34  |  Liver        |     A      |

// Menu not shown in sample i/o

Please select an option: AO

Please enter the organ donor name: John Doe
Please enter the organs John Doe is donating: Lungs
Please enter the blood type of John Doe: O
Please enter the age of John Doe: 72

The organ donor with ID 7 was successfully added to the donor list!

// Menu not shown in sample i/o

Please select an option: LR

Index | Recipient Name     | Age | Organ Needed  | Blood Type | Donor IDs
=========================================================================
   0  | Kevin Malone       | 45  |  Heart        |     O      | 0
   1  | Stanley Hudson     | 40  |  Heart        |     AB     | 0, 4
   2  | Meredith Palmer    | 38  |  Liver        |     B      | 2, 5, 6
   3  | Creed Bratton      | 70  |  Lungs        |     O      | 7
   4  | Jim Halpert        | 34  |  Liver        |     A      |

// Menu not shown in sample i/o

Please select an option: LO

Index | Donor Name         | Age | Organ Donated | Blood Type | Recipient IDs
=============================================================================
   0  | Anonymous          | 43  | Heart         |    O       | 0, 1
   1  | Pam Beesly         | 28  | Kidney        |    O       |
   2  | Andy Bernard       | 38  | Liver         |    B       | 2
   3  | Oscar Nunez        | 33  | Liver         |    AB      |
   4  | Anonymous          | 21  | Heart         |    B       | 0
   5  | Michael Scott      | 47  | Liver         |    B       | 2
   6  | Gabe Lewis         | 26  | Liver         |    B       | 2
   7  | John Doe           | 72  | Lungs         |    O       | 3

// Menu not shown in sample i/o

Please select an option: AR

Please enter new recipient's name: John Smith
Please enter the recipient's blood type: B
Please enter the recipient's age: 26
Please enter the organ needed: Kidney

John Smith is now on the organ transplant waitlist!

// Menu not shown in sample i/o

Please select an option: LR

Index | Recipient Name     | Age | Organ Needed  | Blood Type | Donor IDs
==========================================================================
   0  | Kevin Malone       | 45  |  Heart        |    O       | 0
   1  | Stanley Hudson     | 40  |  Heart        |    AB      | 0, 4
   2  | Meredith Palmer    | 38  |  Liver        |    B       | 2, 5, 6
   3  | Creed Bratton      | 70  |  Lungs        |    O       | 7
   4  | Jim Halpert        | 34  |  Liver        |    A       |
   5  | John Smith         | 26  |  Kidney       |    B       | 1

// Menu not shown in sample i/o
```

```
Please select an option: LO

Index | Donor Name          | Age | Organ Donated | Blood Type | Recipient IDs
==============================================================================
   0  | Anonymous           | 43  | Heart         |     O      | 0, 1
   1  | Pam Beesly          | 28  | Kidney        |     O      | 5
   2  | Andy Bernard        | 38  | Liver         |     B      | 2
   3  | Oscar Nunez         | 33  | Liver         |     AB     |
   4  | Anonymous           | 21  | Heart         |     B      | 1
   5  | Michael Scott       | 47  | Liver         |     B      | 2
   6  | Gabe Lewis          | 26  | Liver         |     B      | 2
   7  | John Doe            | 72  | Lungs         |     O      | 3

// Menu not shown in sample i/o

Please select an option: RO

Please enter the name of the organ donor to remove: Pam Beesly

Pam Beesly was removed from the organ donor list.

// Menu not shown in sample i/o

Please select an option: LO

Index | Donor Name          | Age | Organ Donated | Blood Type | Recipient IDs
==============================================================================
   0  | Anonymous           | 43  | Heart         |     O      | 0, 1
   1  | Andy Bernard        | 38  | Liver         |     B      | 2
   2  | Oscar Nunez         | 33  | Liver         |     AB     |
   3  | Anonymous           | 21  | Heart         |     B      | 1
   4  | Michael Scott       | 47  | Liver         |     B      | 2
   5  | Gabe Lewis          | 26  | Liver         |     B      | 2
   6  | John Doe            | 72  | Lungs         |     O      | 3

// Menu not shown in sample i/o

Please select an option: RR

Please enter the name of the recipient to remove: Jim Halpert

Jim Halpert was removed from the organ transplant waitlist.

// Menu not shown in sample i/o

Please select an option: LR

Index | Recipient Name      | Age | Organ needed | Blood Type | Donor IDs
==========================================================================
   0  | Kevin Malone        | 45  |  Heart       |      O     | 0
   1  | Stanley Hudson      | 40  |  Heart       |      AB    | 0, 3
   2  | Meredith Palmer     | 38  |  Liver       |      B     | 1, 4, 5
   3  | Creed Bratton       | 70  |  Lungs       |      O     | 6
   4  | John Smith          | 26  |  Kidney      |      B     |

// Menu not shown in sample i/o

Please select an option: SR

    (I) Sort by ID
    (N) Sort by Number of Donors
    (B) Sort by Blood Type
    (O) Sort by Organ Needed
    (Q) Back to Main Menu

Please select an option: N

Index | Recipient Name      | Age | Organ needed | Blood Type | Donor IDs
==========================================================================
   4  | John Smith          | 26  |  Kidney      |      B     |
   0  | Kevin Malone        | 45  |  Heart       |      O     | 0
   1  | Stanley Hudson      | 40  |  Heart       |      AB    | 0, 3
   3  | Creed Bratton       | 70  |  Lungs       |      O     | 6
   2  | Meredith Palmer     | 38  |  Liver       |      B     | 1, 4, 5

    (I) Sort by ID
    (N) Sort by Number of Donors
    (B) Sort by Blood Type
    (O) Sort by Organ Needed
    (Q) Back to Main Menu

Please select an option: B
```

```
Index | Recipient Name     | Age | Organ needed | Blood Type | Donor IDs
========================================================================
   1  | Stanley Hudson     | 40  |  Heart       |     AB     | 0, 3
   2  | Meredith Palmer    | 38  |  Liver       |     B      | 1, 4, 5
   4  | John Smith         | 26  |  Kidney      |     B      |
   0  | Kevin Malone       | 45  |  Heart       |     O      | 0
   3  | Creed Bratton      | 70  |  Lungs       |     O      | 6

    (I) Sort by ID
    (N) Sort by Number of Donors
    (B) Sort by Blood Type
    (O) Sort by Organ Needed
    (Q) Back to Main Menu

Please select an option: Q // When exiting the sorting submenu, you should revert your data structure to its initial, unsorted state

Returning to main menu.

// Menu not shown in sample i/o

Please select an option: LR

Index | Recipient Name     | Age | Organ needed | Blood Type | Donor IDs
========================================================================
   0  | Kevin Malone       | 45  |  Heart       |     O      | 0
   1  | Stanley Hudson     | 40  |  Heart       |     AB     | 0, 3
   2  | Meredith Palmer    | 38  |  Liver       |     B      | 1, 4, 5
   3  | Creed Bratton      | 70  |  Lungs       |     O      | 6
   4  | John Smith         | 26  |  Kidney      |     B      |

// Menu not shown in sample i/o

Please select an option: Q

Writing data to transplant.obj...

// Program is started again

Loading data from transplant.obj...

Menu:
    (LR) - List all recipients
    (LO) - List all donors
    (AO) - Add new donor
    (AR) - Add new recipient
    (RO) - Remove donor
    (RR) - Remove recipient
    (SR) - Sort recipients
    (SO) - Sort donors
    (Q) - Quit

Please select an option: LR

Index | Recipient Name     | Age | Organ needed | Blood Type | Donor IDs
========================================================================
   0  | Kevin Malone       | 45  |  Heart       |     O      | 0
   1  | Stanley Hudson     | 40  |  Heart       |     AB     | 0, 3
   2  | Meredith Palmer    | 38  |  Liver       |     B      | 1, 4, 5
   3  | Creed Bratton      | 70  |  Lungs       |     O      | 6
   4  | John Smith         | 26  |  Kidney      |     B      |

// Menu not shown in sample i/o

Please select an option: SO

    (I) Sort by ID
    (N) Sort by Number of Recipients
    (B) Sort by Blood Type
    (O) Sort by Organ Donated
    (Q) Back to Main Menu

Please select an option: O

Index | Donor Name         | Age | Organ Donated | Blood Type | Recipient IDs
============================================================================
   0  | Anonymous          | 43  | Heart         |     O      | 0, 1
   3  | Anonymous          | 21  | Heart         |     B      | 1
   1  | Andy Bernard       | 38  | Liver         |     B      | 2
   2  | Oscar Nunez        | 33  | Liver         |     AB     |
   4  | Michael Scott      | 47  | Liver         |     B      | 2
   5  | Gabe Lewis         | 26  | Liver         |     B      | 2
   6  | John Doe           | 72  | Lungs         |     O      | 3
```

```
    (I) Sort by ID
    (N) Sort by Number of Donors
    (B) Sort by Blood Type
    (O) Sort by Organ Needed
    (Q) Back to Main Menu

Please select an option: I

Index | Donor Name           | Age | Organ Donated | Blood Type | Recipient IDs
==============================================================================
   0  | Anonymous            | 43  | Heart         |     O      | 0, 1
   1  | Andy Bernard         | 38  | Liver         |     B      | 2
   2  | Oscar Nunez          | 33  | Liver         |     AB     |
   3  | Anonymous            | 21  | Heart         |     B      | 1
   4  | Michael Scott        | 47  | Liver         |     B      | 2
   5  | Gabe Lewis           | 26  | Liver         |     B      | 2
   6  | John Doe             | 72  | Lungs         |     O      | 3

    (I) Sort by ID
    (N) Sort by Number of Donors
    (B) Sort by Blood Type
    (O) Sort by Organ Needed
    (Q) Back to Main Menu

Please select an option: Q // When exiting the sorting submenu, you should revert your data structure to its initial, unsorted state

Returning to main menu.

// Menu not shown in sample i/o

Please select an option: Q

Writing data to transplant.obj...

Program terminating normally...
```