COMP 2611 – Data Structures

2020/2021 Semester 1

Two-Day Final Assessment

Date Due: *Friday December 18, 2020 @ 11.55 pm*

Description

This assessment requires you to write an application for handling patients who come in for treatment at the emergency department of a certain hospital. The application uses several of the data structures covered in the course. In particular, it uses the data structures listed in the following table:

Data Structure	**Purpose**
Binary search tree	Stores information on patients (the patient "database").
Max-priority queue (based on an underlying max-heap)	Stores the priority of patients waiting to be treated.
Hash table (using linear probing with double hashing)	Stores a list of conditions and the priority that is assigned to each one.
Queue (based on an underlying linked list)	Maintains a list of patients who have already been treated on a particular day.

Table 1: Data Structures Used in 2-Day Assessment

When a patient comes in for treatment, if he/she is in the patient database, a priority is assigned to the patient and the patient's information is inserted in a max priority queue. The priority assigned depends on the nature of the emergency. If the patient is not in the patient database, his/her information is first stored in the patient database and then a priority is assigned to the patient and his/her information is inserted in the max priority queue.

Patients are treated in order of priority. The next patient to be treated is the one with the highest priority in the max priority queue. When a patient is treated, his/her information is removed from the priority queue and inserted in another queue which keeps track of the patients who have already been treated.

The application must provide a menu to facilitate the entering of new patient information in the patient database and to handle patients as they come in for treatment. The menu must also allow the user to view the information stored in the different data structures.

You are supplied with a Dev C++ project containing various `.h` and `.cpp` files. You do not have to modify the project. The code for this assessment must be written in the `BinarySearchTree.cpp`, `HashTable`.cpp, `Main.cpp`, `and MaxPriorityQueue.cpp` files. There is no need to modify the code in the other files supplied.

Structure Definitions

The file, `DataTypes.h`, defines the data types to be used in this assessment. They are listed in Table 2:

Data Type	Description
Condition	A struct containing conditions and priorities which are stored in a hash table.
Patient	A struct containing patient information which is stored in the patient database (binary search tree).
Incident	A struct containing information about a patient to be treated. The max priority queue and the queue of patients already treated contain data of type *Incident*.
BTNode	A node in the binary search tree (which stores data of type *Patient*).
QueueNode	A node in the queue of patients already treated.

Table 2: Data Types

Heap Functions (all provided in MaxHeap.cpp)

Return type	Function and Description
`int`	`parent (int i):` Returns the index of the parent node of node *i* in the heap.
`int`	`leftChild (int i):` Returns the index of the left child of node *i* in the heap.
`int`	`rightChild (int i):` Returns the index of the right child of node *i* in the heap.
`void`	`maxHeapify (Incident A[], int heapSize, int i):` Assuming that the left and right subtrees of node *i* are max heaps, maintains the max heap property starting at node *i*.
`void`	`buildMaxHeap (Incident A[], int lengthA):` Given an array *A* of type *Incident* where *lengthA* is the amount of elements in *A*, converts *A* into a max heap based on the priority of each patient.
`void`	`heapSort (Incident A[], int lengthA):` Given an array *A* of type *Incident* and *lengthA*, which is the amount of elements in *A*, sorts the elements in *A* based on the priority of the incidents.

Priority Queue Functions (to be written in PriorityQueue.cpp)

Return type	Function and Description
`void`	`heapInsert (Incident A[], int heapSize, Incident newIncident):` Inserts *newIncident* in the priority queue stored in array *A* based on its priority.
`Incident`	`heapMaximum (Incident A[], int heapSize):` Returns the highest priority element in the max priority queue stored in array *A*.
`Incident`	`heapExtractMax (Incident A[], int heapSize):` Removes and returns the highest priority element in the max priority queue stored in array *A*.
`void`	`heapIncreasePriority (Incident A[], int heapSize, int i, int newPriority):` *Increases* the value of element *i's* priority to the new value, *newPriority*, where *newPriority* ≥ element *i's* current priority.

Binary Search Tree Functions (all provided in BinarySearchTree.cpp except getBST)

Return type	Function and Description
`BTNode *`	`createBTNode (Patient newPatient):` Creates a node in the BST and stores the *newPatient* struct in the node. Returns the address of the node created.
`bool`	`containsBST (BTNode * root, int patientID):` Returns *true* if *patientID* is present in the BST and *false*, otherwise.
`BTNode *`	`getBST (BTNode * root, int patientID):` Returns the node of the BST containing *patientID* if it is present and NULL, otherwise.
`BTNode *`	`insertBST (BTNode * root, Patient newPatient):` Inserts the *newPatient* struct in the BST. The key is *patientID*. Returns the address of the node inserted.
`BTNode *`	`treeMinimum (BTNode * root):` Given the root of the BST, returns the address of the node in the BST with the smallest key.
`BTNode *`	`inOrderSuccessor (BTNode * node):` Given a node in the BST, returns the address of the inorder successor of the node. If it does not exist, returns NULL.

Hash Table Functions (to be written in HashTable.cpp)

Return type	Function and Description
`HashTable *`	`initHT (int sizeTable):` Creates a new hash table with the given size, initializes the hash table, and returns the address of the hash table created.
`bool`	`bool containsHT (HashTable * ht, string key):` Returns *true* if the hash table contains *key* and *false*, otherwise.
`void`	`insertHT (HashTable * ht, Condition c):` Inserts an element into the hash table (a struct of type *Condition*).

Application (To be written in Main.cpp)

You are required to write an application which, on startup, reads Patient information from the file, `Patient.txt`, and stores the information in a binary search tree. The key is the patient ID. Each line of data in the `Patient.txt` file contains the following information:

```
Patient Id          Patient Name (no spaces)         Patient Telephone Contact
```

Next, the application reads Incident information from the file, `Incident.txt`, and stores each Incident in a priority queue based on the priority of the Incident. Each line of data in the `Incident.txt` file contains the following information:

```
Patient Id          Condition (no spaces)            Priority
```

Next, your application should read Condition and priority information from the file, `Priority.txt`. Each line of data in the file contains the following information:

```
Condition           Priority
```

The priority information should be stored in a hash table where the key is the Condition (a string). Linear probing with double hashing should to be used to resolve collisions.

You should create your own data for `Patient.txt`, `Incident.txt`, and `Priority.txt`.

After reading the files, the application should provide a menu from which various operations are performed. The operations are performed by calling the functions written for the different data structures.

The following is the menu that should be displayed:

```
Patient Management System
--------------------------------------------------------
1.   Add New Patient to Patient Database
2.   Add Patient to Priority Queue
3.   Increase Patient Priority
4.   Display Information on Next Patient to be Treated
5.   Treat Highest Priority Patient
6.   Display List of Patients Currently Awaiting Treatment
7.   Display List of Patients Already Treated
8.   Display List of Patients in Patient Database
9.   Add New Condition and Priority to Hash Table
10.  Quit

Please enter an option:
```

When an option is selected, the appropriate action must be taken, after which the menu is re-displayed. The following is a description of each option.

Option 1:

When this option is selected, the program should allow the user to enter the patient ID, name, and telephone contact for a new patient. The data must be stored in the binary search tree containing patient information.

Option 2:

This option is selected when a patient comes to the emergency department for treatment. The program should allow the user to enter the patient ID, condition, and priority of the patient. A check is first made to determine if the patient in stored in the patient database. If so, the hash table of conditions is consulted to determine the priority that will be assigned to the patient. The patient information is then stored in the priority queue based on his/her assigned priority. If the patient is not stored in the patient database, the user is requested to enter the new patient using Option 1.

Option 3:

This option allows the priority of a given patient on the priority queue to be increased. A check must first be made to determine if the patient is currently on the priority queue.

Option 4:

When this option is selected, the program should display information on the highest priority patient on the queue (if there is one).

Option 5:

When this option is selected, the program should remove the highest priority patient from the queue (if there is one). At the same time, the patient is added to an ‘ordinary’ queue containing a list of patients already treated.

Option 6:

When this option is selected, the program should display the name, condition, and priority of all the patients waiting to be treated, in order of priority.

Option 7:

When this option is selected, the program should display the name, condition, and priority of all the patients who have already been treated, in the order in which they were treated.

Option 8:

When this option is selected, the program should display information on all the patients stored in the patient database, in order of patient ID. **Only** the functions given in `BinarySearchTree.cpp` should be used to extract the patient information from the binary search tree.

Option 9:

When this option is selected, a new condition and priority are added to the hash table of conditions and priorities.

Note for Options 6 and 7

You may have to copy or alter the data in one or more data structures to achieve the functionality required. If data is altered, the data structure must be restored to its original state after displaying the information requested.

What You Have to Submit

(1) **All** the files that were provided in the Dev-C++ project with your modifications in `BinarySearchTree.cpp`, `HashTable`.cpp, `Main.cpp`, and `MaxPriorityQueue.cpp`.

(2) The three data files, `Patient.txt`, `Incident.txt`, and `Priority.txt`.

(3) A PDF document stating the extent to which you believe that menu Options 1 through 9 are working correctly. You should also state any issues you encountered in using the supplied Dev-C++ project files and explain how these issues were handled.