Hacettepe University – Computer Engineering
BBM203 Software Practicum I – Fall 2023

PROGRAMMING ASSIGNMENT 4

SAVING DR. ELARA: MISSION RESCUE

Submission Deadline: 29/12/2023 at 23:59:59
Over submit.cs.hacettepe.edu.tr

Topics: Binary Search Trees, Red-Black Trees, KD-trees, kNN classification, Recursion, Dynamic Memory Allocation, File I/O

Course Instructors: Assoc. Prof. Dr. Adnan ÖZSOY, Asst. Prof. Dr. Engin DEMİR, Assoc. Prof. Dr. Hacer YALIM KELEŞ
TAs: Alperen ÇAKIN, Ardan YILMAZ, Dr. Selma DİLEK (all TAs have contributed to this assignment)
Programming Language: C++11 - **You MUST USE this starter code**
Due Date: **Friday, 29/12/2023 (23:59:59)**

Saving Dr. Elara: Mission Rescue

Following the groundbreaking decryption of the Celestial Tapestry in *Programming Assignment 1*, a glimmer of hope emerged in the search for the enigmatic Dr. Elara. The tapestry, once thought to be a mere cosmic anomaly, revealed itself as a beacon, guiding us to the whereabouts of Dr. Elara, who vanished while exploring the uncharted depths of space.

The tapestry's hidden messages, a cryptic blend of celestial coordinates and obscure references to distant planets, have led to the formulation of a daring rescue operation: *Mission Rescue*. This high-stakes venture, spearheaded by the same brilliant team of HUBBM and HUAIN students who unraveled the tapestry's secrets, aims to traverse the perilous expanses of space to retrieve Dr. Elara from where she is believed to be stranded.

The path to Dr. Elara is fraught with challenges that demand not only courage but unparalleled scientific and computational expertise. To navigate these unknown realms, three critical tasks have been identified, forming the backbone of the mission's strategy: *planetary classification system*, *space sector mapping*, and *space sector mapping optimization*.

Mission Rescue is not just a simple retrieval mission; it is a journey into the unknown, a testament to human ingenuity and the relentless pursuit of knowledge. As the team embarks on this perilous voyage, they carry not only the hope of finding Dr. Elara but also the aspirations of unraveling the mysteries that lie beyond our known universe.

1 Task 1: Space Sector Mapping

The vastness of uncharted space requires a comprehensive mapping system. In this part, you are tasked with implementing Binary Search Tree (BST) algorithms to map space sectors along exploration routes. Your goal is to identify the path to the final destination sector, where Dr. Elara is believed to be stranded on one of the habitable planets.

As the journey unfolds, new sectors may be discovered that need to be integrated into the space map, while the sectors identified as dangerous must be removed from the map. This dynamic process will enable the explorers to adjust the mission's trajectory in real-time, ensuring a strategic and responsive approach to the ever-evolving landscape of space exploration.

1.1 Input File and Sector Map Initialization

Space sectors are represented by their position relative to the Earth's position on a 3D coordinates (X, Y, Z) plane. An input file containing the coordinates of the already discovered space **sectors**, structured as a `DAT` file, will be given as the *first command line argument*. Your task is to parse this file within your program to set up the initial `BST` that will represent the sector map within the `SpaceSectorBST` class. The member pointer variable `Sector *root` must be set to point to the root node of this tree. The content of a sample input file given as `sectors.dat` is illustrated on the right.

```
X,Y,Z
0,0,0
10,20,30
-5,-15,10
25,5,0
-10,-20,-30
5,15,-10
-5,15,10
-25,-5,0
30,40,50
15,-25,35
0,30,-40
-10,-20,30
15,-20,35
```

Sector nodes should be inserted into the BST based on their (X, Y, Z) coordinates as follows:

- **Primary, Secondary, and Tertiary Sorting Criteria**: The primary sorting criterion is the X-coordinate. If two sectors have the same X-coordinate, the Y-coordinate becomes the secondary sorting criterion. If both X and Y coordinates are the same, the Z-coordinate is used as the tertiary sorting criterion.
- **Insertion Process**: Start by comparing the X-coordinates of sectors. For sectors with the same X-coordinate, compare their Y-coordinates. For sectors with identical X and Y coordinates, compare their Z-coordinates.
- **Tree Structure**: A node's left child has a smaller X-coordinate or, in case of a tie, a smaller Y-coordinate, or, if both are tied, a smaller Z-coordinate. A node's right child has a greater or equal X-coordinate (and then Y, Z are considered similarly in case of ties).

You are expected to implement the node insertion recursively. In a recursive approach, the tree is traversed from the root down to the appropriate null link, and the new node is inserted at that point. The recursive function will:

- Take the current node and its coordinates as parameters.
- Begin the insertion with the root node of the BST. If the BST is empty (root is `nullptr`), the new node becomes the root. Otherwise, compare the coordinates as explained above. If the sorting criteria require inserting the sector to the left, proceed to the left child. Otherwise, proceed to the right child.

- Recursive Call: Make a recursive call to the insert function with the left or right child based on the comparison. If the child is `nullptr`, create a new node with the given distance and attach it at this position.
- Return the current node after each recursive call to maintain the tree structure. This ensures that the root of the subtree (or the overall tree) is updated correctly after the insertion.
- The node returned from the initial call of the function (starting at the root) is assigned as the new root of the BST.

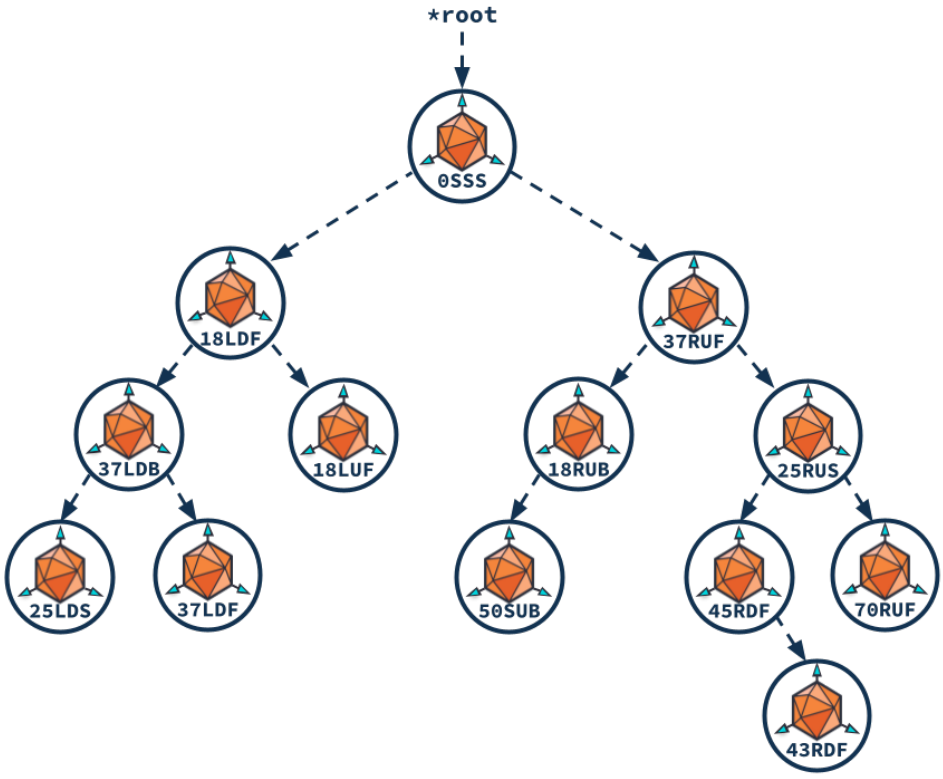
Each sector can be uniquely identified within the BST map by its **sector code** that is calculated based on its (X, Y, Z) coordinates and its distance from the Earth. To calculate the sector distance in Astro Units (AUs) from the Earth based on their coordinates, you should use the Euclidean distance formula in three dimensions: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$. The Earth is considered to have the origin point of $(0, 0, 0)$.

Sector Code Generation Logic:

- **Distance Component**: The first part of the sector code is derived from the sector's distance from Earth, truncated to integer (you can think of it as the floor operation). ← [updated]
- **Coordinate Components**: The next parts of the code are determined by the sector's X, Y, and Z coordinates.
  - For each coordinate, if the coordinate value is 0 (same as the Earth's), append the letter '**S**' to the code, signifying 'Same' in that dimension.
  - If the coordinate value is positive, append the respective directional letter: '**R**' for Right (X), '**U**' for Up (Y), and '**F**' for Forward (Z).
  - If the coordinate value is negative, append the respective directional letter: '**L**' for Left (X), '**D**' for Down (Y), and '**B**' for Backward (Z).

An example: For a sector located at coordinates $(10, -5, 20)$ and 22.91 AUs away from Earth, the distance component is **22**. The coordinate components are `R` for X (positive), `D` for Y (negative), and `F` for Z (positive). So, the resulting sector code is **22RDF**. ← [updated]

After reading the sample input provided in the given `sectors.dat` file, and inserting the sector nodes into the BST in order they appear in the input file, while adhering to the sorting criteria and calculating the sector codes correctly, you should obtain the sector map shown on the right.

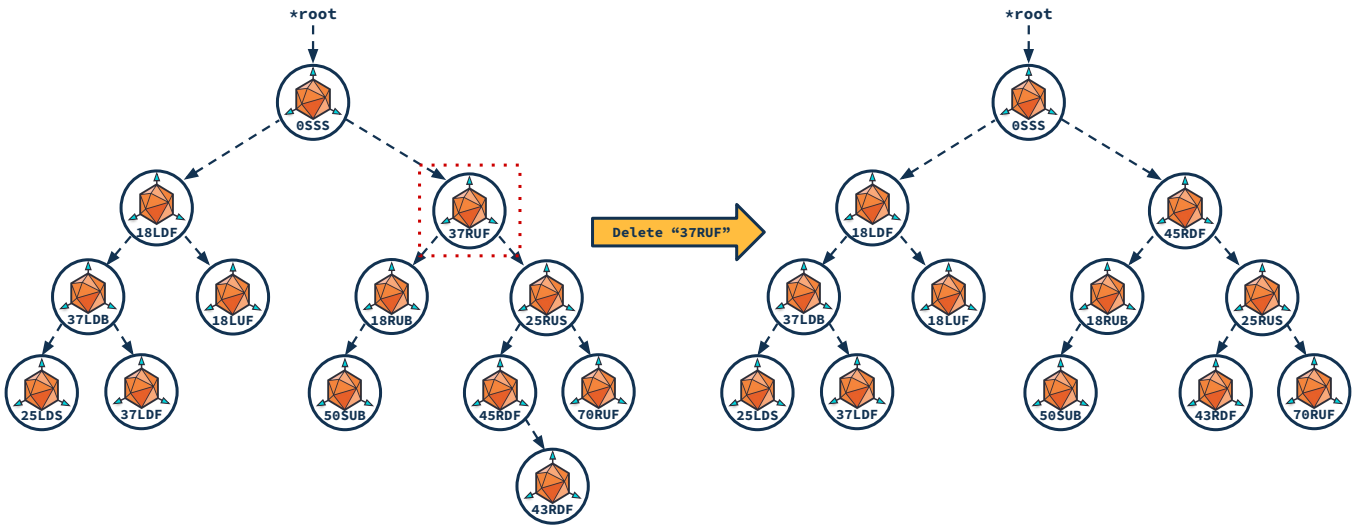


1.2 Deleting Dangerous Sectors from the Sector Map

When a sector is deemed dangerous, it must immediately be removed from the map. A dangerous sector will be identified by its `sector_code`. **You are expected to implement the sector node deletion recursively**. To implement deleting a sector node from the sector tree map, you'll need to follow the standard procedure for deleting a node from a binary search tree. This involves finding the node with the matching `sector_code`, then handling three cases if found:

- Node with no children (leaf node): Simply remove the node.
- Node with one child: Replace the node with its child, and then delete the node.
- Node with two children: Find the node's **in-order successor** (smallest node in the right subtree), swap it with the node, and then delete the node.

For instance, if the sector **37RUF** is identified as dangerous, the resulting tree, shown on the right in the figure below, illustrates the sector tree map after the removal of this sector.



1.3 Finding a Path to Dr. Elara

In order to be able to search through the sector tree map, you need to implement three BST traversal algorithms: **inorder**, **preorder**, and **postorder**. In a BST, inorder traversal retrieves elements in sorted order, preorder traversal is useful for creating prefix expressions and operations involving processing the root before the children, while postorder traversal is commonly employed for deleting nodes and processing children before the root. Below, the expected traversal outputs are given for the initial sample tree:

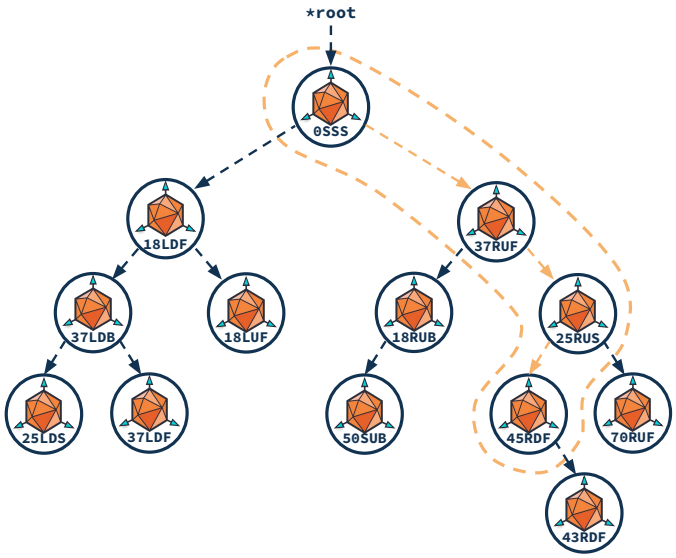
```
Space sectors inorder traversal:
25LDS
37LDB
37LDF
18LDF
18LUF
0SSS
50SUB
18RUB
37RUF
45RDF
43RDF
25RUS
70RUF
```

```
Space sectors preorder traversal:
0SSS
18LDF
37LDB
25LDS
37LDF
18LUF
37RUF
18RUB
50SUB
25RUS
45RDF
43RDF
70RUF
```

```
Space sectors postorder traversal:
25LDS
37LDF
37LDB
18LUF
18LDF
50SUB
18RUB
43RDF
45RDF
70RUF
25RUS
37RUF
0SSS
```

In Programming Assignment 1, you successfully decoded a hidden message within the Celestial Tapestry. This message revealed a sector code, believed to be a secret communication from Dr. Elara, notifying us about her location in the cosmos. It's now imperative that we organize a rescue mission to ensure Dr. Elara's safe return to Earth.

Your mission in this assignment is to navigate the space sector tree map to determine the exact path needed to reach the sector where Dr. Elara is stranded. Envision space as interconnected by wormholes, linking sectors directly in a manner reflected by the child links in our sector tree map. Starting from Earth's sector, the rescue team will use these inter-sector connections to travel, or perform 'space jumps', from one sector to another, ultimately reaching Dr. Elara's location. You are tasked with implementing a function that returns a vector of pointers to Sector nodes, such that, following these pointers in sequence will guide the rescue team from Earth to Dr. Elara's sector. For instance, if based on the decoded message, we were informed that Dr. Elara is in sector **45RDF**. Using the initial sample tree structure (before any deletions), the expected path (illustrated on the right) should be printed like this:



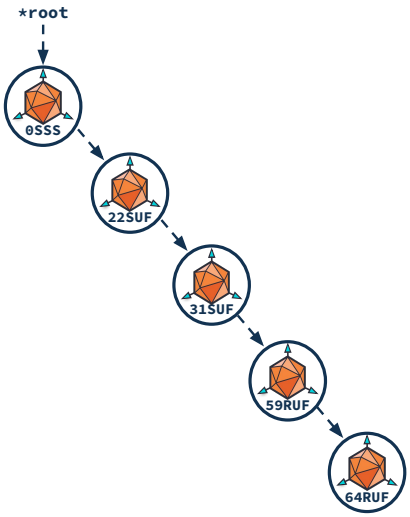

```
The stellar path to Dr. Elara: 0SSS->37RUF->25RUS->45RDF
```

In this example, the expected path to the intended sector necessitates three space jumps to reach the destination from the Earth.

1.4 Challenges of Using Unbalanced BSTs in Space Sector Mapping

Now, imagine that the sector input file was given such that the sectors are already sorted based on the insertion criteria. For example, let the sample `sectors_sorted.dat` file be structured as shown on the right.

```
X,Y,Z
0,0,0
0,10,20
0,10,30
10,30,50
10,40,50
```



Using the approach we employed to generate the BST map could lead to a completely imbalanced tree as illustrated on the left. This imbalance has significant drawbacks, notably prolonged search times. More critically, it could compel the rescue team to undertake a considerably lengthier stellar journey to reach the destination sector where Dr. Elara is stranded. For instance, in this particular map, if the destination sector is **59RUF**, the team would need to make three space jumps to get there. In contrast, if the tree were balanced, they might only need one or two jumps. The efficiency of the path is directly affected by the tree's structure, emphasizing the importance of a balanced tree in optimizing the rescue mission.

For this reason, in the next task of this assignment, you are expected to implement a better, balanced version of the sector tree map.

2 Task 2: Optimizing Space Sector Mapping

In this task, you are expected to use the same input `DAT` file, given as the *first command line argument*, parse the sectors by the given coordinates to implement a new sector tree map, but this time, you must use a Left-Leaning Red-Black Tree (LLRBT) data structure instead of BST. The choice of an LLRBT is crucial for this task due to its self-balancing properties.

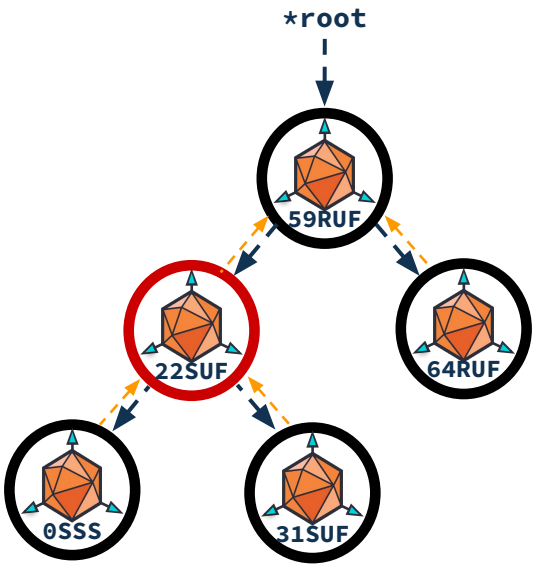
Node Insertion into the Left-Leaning Red-Black Tree (LLRBT): In this task, you are expected to implement recursive node insertion into the LLRBT. This tree structure is a variation of the standard Red-Black Tree, with a specific emphasis on ensuring that red links lean left. The recursive function for inserting a new sector node into the LLRBT will follow these steps:

- *Parameter Handling:* Take the current node and its coordinates as parameters.
- *Initiation:* Start the insertion with the root node of the LLRBT. If the tree is empty (root is `nullptr`), the new node becomes the root and is colored `BLACK`. Otherwise, proceed based on the sector coordinates. Insert to the left child if criteria dictate, otherwise to the right.
- *Recursive Insertion:* Make a recursive call to the insert function with the appropriate child (left or right) based on the comparison. If reaching a `nullptr` position, create a new red node with the given coordinates.
- *Fixing Violations:* After insertion, fix any violations of the LLRBT properties. This includes enforcing left-leaning red links, balancing consecutive red links, and ensuring every path from a node to descendant leaves contains the same number of black nodes. This is achieved through rotations (left or right as needed) and color flips.
- *Structure Maintenance:* Return the current node after each recursive call. This step is crucial as the subtree's root (or the overall tree) might change due to rotations during the fixing process.
- *Root Assignment:* The node returned from the initial function call (starting at the root) becomes the new root of the LLRBT.

Please note, the insertion process in an LLRBT differs significantly from both a standard Binary Search Tree and a traditional Red-Black Tree. It requires specific operations such as left-leaning rotations and color flips to maintain the unique properties of LLRBTs.

Remember the example from Section 1.4 where we obtained a completely imbalanced, right-leaning tree? In that particular case, the destination sector was **59RUF**, and the rescue team needed to make three space jumps to reach Dr. Elara.

In contrast, by using a Red-Black Tree instead of a Binary Search Tree, our new space sector map is balanced (see the figure on the right), and the rescue team will only need to perform two space jumps instead of three. Note that the parent pointers are represented as yellow arrows, while the node color is shown as black or red node outlines. The efficiency of the rescue path is directly affected by the tree's structure, emphasizing the importance of a balanced tree in optimizing the rescue mission.



2.1 Finding an Optimized Path to Dr. Elara

In order to be able to search through the sector tree map, you can use the same three BST traversal algorithms: **inorder**, **preorder**, and **postorder**, implemented in the previous task. Below, the expected traversal outputs are given for the obtained sample tree:

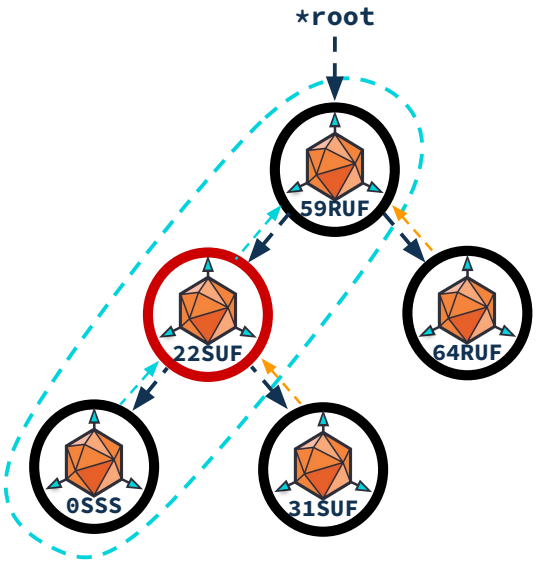
```
Space sectors inorder traversal:
BLACK sector: 0SSS
RED sector: 22SUF
BLACK sector: 31SUF
BLACK sector: 59RUF
BLACK sector: 64RUF
```

```
Space sectors preorder traversal:
BLACK sector: 59RUF
RED sector: 22SUF
BLACK sector: 0SSS
BLACK sector: 31SUF
BLACK sector: 64RUF
```

```
Space sectors postorder traversal:
BLACK sector: 0SSS
BLACK sector: 31SUF
RED sector: 22SUF
BLACK sector: 64RUF
BLACK sector: 59RUF
```

Note that this time, the color of the sector in the tree is also displayed.

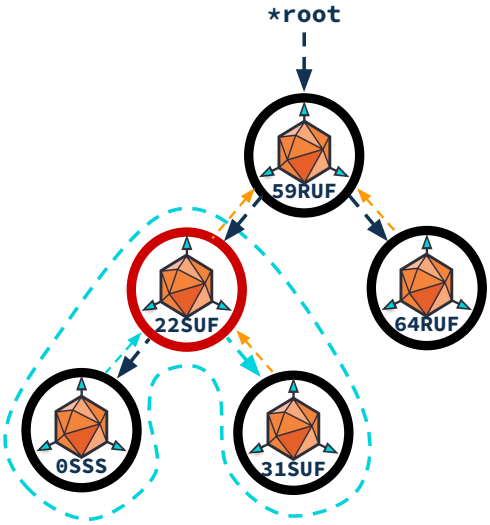
Your mission in this task is to navigate the space sector tree map to determine the exact path needed to reach the sector where Dr. Elara is stranded, noting that the sector where the Earth is located may not be the root sector node any longer. Sectors are again connected by two-way wormholes, reflected by the child and parent links in our new sector tree map. Starting from Earth's sector, the rescue team will use these inter-sector connections to perform 'space jumps' from one sector to another, ultimately reaching Dr. Elara's location. You are tasked with implementing a function that returns a vector of pointers to Sector nodes, such that, following these pointers in sequence will guide the rescue team from Earth to Dr. Elara's sector. For instance, if based on the decoded message, we were informed that Dr. Elara is in sector **59RUF**. Using the obtained sample tree structure, the expected path (illustrated on the right) should be printed like this:




```
The stellar path to Dr. Elara: 0SSS->22SUF->59RUF
```

In this example, the expected path to the intended sector necessitates two space jumps to reach the destination from the Earth.

In the second example illustrated on the right, we assume that that Dr. Elara is in sector **31SUF**, based on the decoded message. Using the obtained sample tree structure, the expected path should be printed like this:




```
The stellar path to Dr. Elara: 0SSS->22SUF->31SUF
```

Via this example, we observe that the path may or may not visit the root of our tree sector map. Another important note: **make sure that there are no duplicate Sector nodes in the path!** If a path cannot be found, the expected output is:

```
A path to Dr. Elara could not be found.
```

4 Assignment Implementation Tasks and Requirements

In this section, we outline the classes and functions you are required to implement. As aspiring engineers, it's crucial to base your code on the provided starter code that offers a predetermined class structure. This ensures code clarity, proper encapsulation, and facilitates unit testing. It's imperative that you do not alter the names or signatures of functions provided in the template files. Likewise, refrain from renaming or changing the types of the specified member variables. Other than that, you're free to introduce any additional functions or variables as needed.

`Sector` Class

- **Constructor**:
  `Sector(int x, int y, int z)`

  - Initializes a sector based on the given coordinates. Calculates the sector's distance from the Earth and generates its unique sector code.
- **Operators** $=$, $==$, $!=$: Overloading the assignment and comparison operators.
- **Destructor**:
  `~Sector()`

  - Frees any dynamically allocated memory if necessary.

`SpaceSectorBST` Class

- **Constructor**:
  `SpaceSectorBST()`

  - Initializes a space sector BST instance with a `nullptr root` (already given).
- **Functions**:
  `void readSectorsFromFile(const std::string& filename)`

  - Reads sectors from the given input file and inserts them into the space sector BST map according to the coordinates-based comparison criteria.

  `void insertSectorByCoordinates(int x, int y, int z)`

  - Instantiates and inserts a new sector into the space sector BST map according to the coordinates-based comparison criteria.

  `void deleteSector(const std::string& sector_code)`

  - Deletes the sector, given by its sector code, from the space sector BST map.

  `void displaySectorsInOrder()`

  - Traverses the space sector BST map in-order and prints the sectors to `STDOUT` in the given format.

  `void displaySectorsPreOrder()`

  - Traverses the space sector BST map in pre-order traversal and prints the sectors to `STDOUT` in the given format.

`void displaySectorsPostOrder()`

- Traverses the space sector BST map post-order traversal and prints the sectors to `STDOUT` in the given format.

`std::vector<Sector*> getStellarPath(const std::string& sector_code)`

- Finds the path from the Earth to the destination sector given by the `sector_code` and returns this path as a vector or `Sector` pointers in the path.

`void printStellarPath(const std::vector<Sector*>& path)`

- Prints the stellar path obtained from the `getStellarPath()` function to `STDOUT` in the given format.

- **Destructor**:
  `~SpaceSectorBST()`

  - Frees any dynamically allocated memory if necessary.

`SpaceSectorLLRBT` Class

- **Constructor**:
  `SpaceSectorLLRBT()`

  - Initializes a space sector BST instance with a `nullptr root` (already given).

- **Functions**:
  `void readSectorsFromFile(const std::string& filename)`

  - Reads sectors from the given input file and inserts them into the space sector LLRBT map according to the coordinates-based comparison criteria.

  `void insertSectorByCoordinates(int x, int y, int z)`

  - Instantiates and inserts a new sector into the space sector LLRBT map according to the coordinates-based comparison criteria.

  > The instructions given for the `displaySectorsInOrder()`, `displaySectorsPreOrder()`, `displaySectorsPostOrder()`, `getStellarPath()`, and `printStellarPath()` functions of the `SpaceSectorBST` class above, are also applicable for this class.

- **Destructor**:
  `~SpaceSectorLLRBT()`

  - Frees any dynamically allocated memory if necessary.

Bonus Part Classes

> Please refer to the comments inside the relevant classes starter code for details.

Must-Use Starter Codes

You MUST use **this starter (template) code**. All headers and classes should be placed directly inside your **zip** archive.

Grading Policy - You May Earn a Total of 120 pts.

- No memory leaks and errors: 10%
  - No memory leaks: 5%
  - No memory errors: 5%
- Implementation of the tasks: 80%
  - Task 1 - Space Sector Mapping: 40%
    - Reading input and properly implementing the corresponding BST structure with node insertions: 15%
    - Proper implementation of deletion from the BST: 10%
    - Proper implementation of BST traversals: 5%
    - Proper implementation of pathfinding: 10%
  - Task 2 - Optimizing Space Sector Mapping: 40%
    - Reading input and properly implementing the corresponding LLRBT structure with node insertions: 25%
    - Proper implementation of pathfinding: 15%
- BONUS Task 3 - Planetary Classification System: 20%
  - Reading the input file and building the KD-Tree: 10%
  - Searching the KD-Tree to extract the nearest neighbors and correctly classify a planet: 10%
- Output tests: 10%

Important Notes

- Do not miss the deadline: **Friday, 29.12.2023 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/fall2023/bbm203), and you are supposed to be aware of everything discussed on Piazza.
- You must test your code via **Tur^{3}Bo Grader** https://test-grader.cs.hacettepe.edu.tr/ (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - **b<studentID>.zip**
    - KDT_Node.h <FILE>
    - KD_Tree.cpp <FILE>
    - KD_Tree.h <FILE>
    - kNN.cpp <FILE>
    - kNN.h <FILE>
    - kNN_Data.h <FILE>
    - kNN_DAT_Parser.h <FILE>

    * Sector.cpp <FILE>
    * Sector.h <FILE>
    * SpaceSectorBST.cpp <FILE>
    * SpaceSectorBST.h <FILE>
    * SpaceSectorLLRBT.cpp <FILE>
    * SpaceSectorLLRBT.h <FILE>
- **You MUST use this starter code**. All classes should be placed directly in your **zip** archive.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).

Build and Run Configuration

Here is an example of how your code will be compiled (note that instead of `main.cpp` we will use our test files):

```
$ g++ -g -std=c++11  *.cpp, *.h -o mission_rescue
```

Or, you can use the provided `Makefile` within the sample input to compile your code:

```
$ make
```

After compilation, you can run the program as follows:

```
$ make
$ ./mission_rescue sectors.dat simple_planet_train.dat
```

Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in the suspension of the involved students.