# CSE 208 Offline 1

## Assignment on Balanced BST

**Deadline: 11:55 PM, 13 June, 2025**

## Problem Specification:

You've been hired as the lead backend engineer at **FlashFrenzy**, a high-energy startup known for its thrilling flash sales of rare tech gadgets, luxury sneakers, and limited-edition collectibles. These sales go live at specific times and attract thousands of users competing to purchase limited-stock items in real-time.
Each product in the system is uniquely identified by an integer ProductID and is associated with a product name (string) and a remaining stock count (integer).
To ensure high performance during rapid updates and queries, your system must support efficient **addition**, **deletion**, **lookup**, and **update** operations. Additionally, since administrators frequently need to view the catalog in sorted order by ProductID, the system must support **sorted traversal** of items.
To meet these requirements, you must implement the system using a **Red-Black Tree** data structure, which maintains balance and guarantees logarithmic time complexity for key operations even under heavy insertions and deletions.

## The inventory has the following methods:

1. **AddItem**:
   Adds a new item with the given ID, name, and stock. If the ProductID already exists, its name and stock are updated with the new values.
2. **BuyItem**:
   Reduces the stock of the given item by the specified quantity. If the stock becomes 0 or less, the item is removed from the system.
3. **CheckItem**:
   Displays the remaining stock for the specified ProductID. If the item does not exist, print Not Available.
4. **ClearInventory**:
   Removes all items from the inventory.
5. **InventorySize**:
   Prints the total number of items currently in the inventory.
6. **Empty:** It checks if the inventory is empty.

7. **ListInventory**:
   Prints the entire product list in **ascending order** of ProductID. Each line should follow the format:

   ProductID ⇒ ProductName (Stock)

## Input Format:

Each line in the input will specify one of the following operations:

1. **AddItem (AI)** followed by an integer and two strings denoting the ProductID, ProductName (in quotes if it contains spaces), and Stock value.

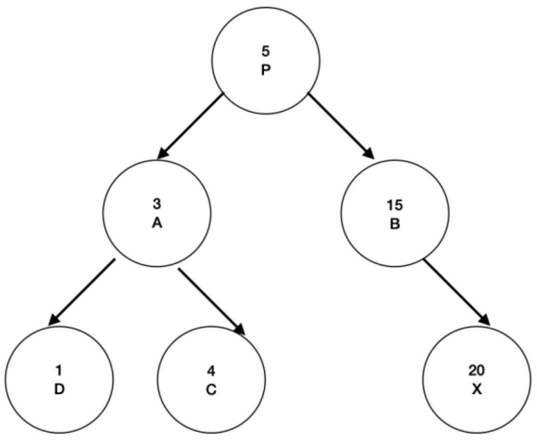
   Example: AI 101 "Smartwatch" 25
2. **BuyItem (BI)** followed by an integer and a value denoting the ProductID and Quantity to buy.

   Example: BI 101 10
3. **CheckItem (CI)** followed by an integer denoting the ProductID to check.

   Example: CI 101
4. **ClearInventory (Clr)** erases all the products from the inventory.
5. **InventorySize (S)** prints the number of items currently in the inventory.
6. **Empty (Em)**
7. **ListInventory (Itr)** performs an in-order traversal of the Red-Black Tree and lists the product details in sorted order based on ProductID.

## Output Format:

Each operation will produce the following output, both in output.txt and on the console:



In the above tree, suppose that the stock of all the products is 5.

1. **AddItem (AI)**: After insertion, the tree will be printed in **nested parentheses format** showing the current structure. You have to print the elements in black or red according to your tree.
   If the item already exists, its details will be updated, and the tree will still be printed.

   Example: (For the above tree)
   5_P(3_A(1_D ,4_C),15_B( ,20_X))
   // Here 5 is Product_ID and P is Product_Name

2. **BuyItem (BI)**: After purchase:
   a. If sufficient stock is available, reduce it.
   b. If stock becomes 0 or less, **remove** the item from the tree.
   c. In both cases, print the updated tree in nested parentheses format.
3. **CheckItem (CI)**:
   a. If the product exists: print "Stock left: X"
   b. Otherwise: print "Not Available"
4. **ClearInventory (Clr)**:
   a. Clears the entire inventory
   b. Output: print "successful" or "unsuccessful"

5. **InventorySize (S)**: print total number of products
6. **Empty (Em):** print "yes" or "no"

7. **ListInventory (ltr)**:
    a. Performs **in-order traversal** and prints each product in sorted order of ProductID. The output of the given tree will be

       1 ⇒ D (5)

       3 ⇒ A (5)

       4 ⇒ C (5)

       5 ⇒ P (5)

       15 ⇒ B (5)

       20 ⇒ X (5)

You have to print the elements in black or red according to your tree.

## Sample I/O:

Use file I/O for this problem. You will take input from input.txt and your output file should be 'output.txt'. You also need to print on the terminal conserving the colour of the nodes in the tree in the following way.

The sample file i/o is here.

| Input | Output |
|---|---|
| Clr | unsuccessful |
| Em | yes |
| AI 10 "Phone" 5 | 10_Phone |
| AI 5 "Mouse" 5 | 10_Phone(5_Mouse,) |
| AI 1 "Charger" 3 | 5_Mouse(1_Charger,10_Phone) |
| AI 15 "Monitor" 7 | 5_Mouse(1_Charger,10_Phone(,15_Monitor)) |
| AI 20 "Laptop" 8 | 5_Mouse(1_Charger,15_Monitor(10_Phone,20_Laptop)) |
| AI 13 "Keyboard" 4 | 5_Mouse(1_Charger,15_Monitor(10_Phone(,13_Keyboard),20_Laptop)) |
| AI 25 "Router" 6 | 5_Mouse(1_Charger,15_Monitor(10_Phone(,13_Keyboard),20_Laptop(,25_Router))) |
| AI 30 "Tablet" 3 | 5_Mouse(1_Charger,15_Monitor(10_Phone(,13_Keyboard),25_Router(20_Laptop,30_Tablet))) |
| AI 28 "Speaker" 2 | 15_Monitor(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),25_Router(20_Laptop,30_Tablet(,28_Speaker))) |
| AI 27 "Camera" 1 | 15_Monitor(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),25_Router(20_Laptop,28_Speaker(27_Camera,30_Tablet))) |
| AI 15 "MonitorPro" 20 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),25_Router(20_Laptop,28_Speaker(27_Camera,30_Tablet))) |
| CI 20 | Stock left: 8 |
| CI 99 | Not available |

| | |
|---|---|
| BI 1 1 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),25_Router(20_Laptop,28_Speaker(27_Camera,30_Tablet))) |
| BI 5 3 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),25_Router(20_Laptop,28_Speaker(27_Camera,30_Tablet))) |
| BI 25 6 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),28_Speaker(20_Laptop(,27_Camera),30_Tablet)) |
| BI 20 8 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),28_Speaker(27_Camera,30_Tablet)) |
| BI 27 1 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),28_Speaker(,30_Tablet)) |
| BI 30 3 | 15_MonitorPro(5_Mouse(1_Charger,10_Phone(,13_Keyboard)),28_Speaker) |
| BI 28 2 | 5_Mouse(1_Charger,13_Keyboard(10_Phone,15_MonitorPro)) |
| BI 10 5 | 5_Mouse(1_Charger,13_Keyboard(,15_MonitorPro)) |
| Em | no |
| S | 4 |
| Itr | 1 => Charger (2)<br>5 => Mouse (2)<br>13 => Keyboard(4)<br>15 => MonitorPro(20) |
| Clr | successful |
| S | 0 |
| Itr | |

## Hints:

1. You may keep the red-black tree implementation in a header file/another source file for reusability purposes and include it.
2. Modularize your code.
3. Using OOP is recommended for better organization of your code.
4. Use **OOP principles**. Recommended structure: [Not a must do thing]
    a. Node class
    b. RedBlackTree class
    c. InventorySystem class
5. Use the following repo to enable **colored console output**:

   a. color-console GitHub

6. You can check this website for red-black tree visualization.

## Submission Instructions:

1. Please DO NOT COPY solutions from anywhere (your friends, seniors, internet, copilot, gpt etc.). Any form of plagiarism (irrespective of source or destination), will result in getting -100% marks in the offline.
2. Rename all the problem solutions according to your student ID. If your ID is 2205XXX,then create a folder named 2205XXX. Afterward, rename the problem as 2205XXX.cpp and move it inside the folder. Create a zip file of that folder. Lastly, submit the zip file.
3. If you can’t give proper answer in the viva, you will get 0, so try to understand all the logics.