MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

INVENTORY MANAGEMENT SYSTEM

Department: Instrumentation & Control Engineering
Branch: Cyber-Physical System
Semester: III
Subject: Data Structures & Algorithms

Submitted By:
Jimit Desai
Rishitaa Prakash
Aryan Satpute

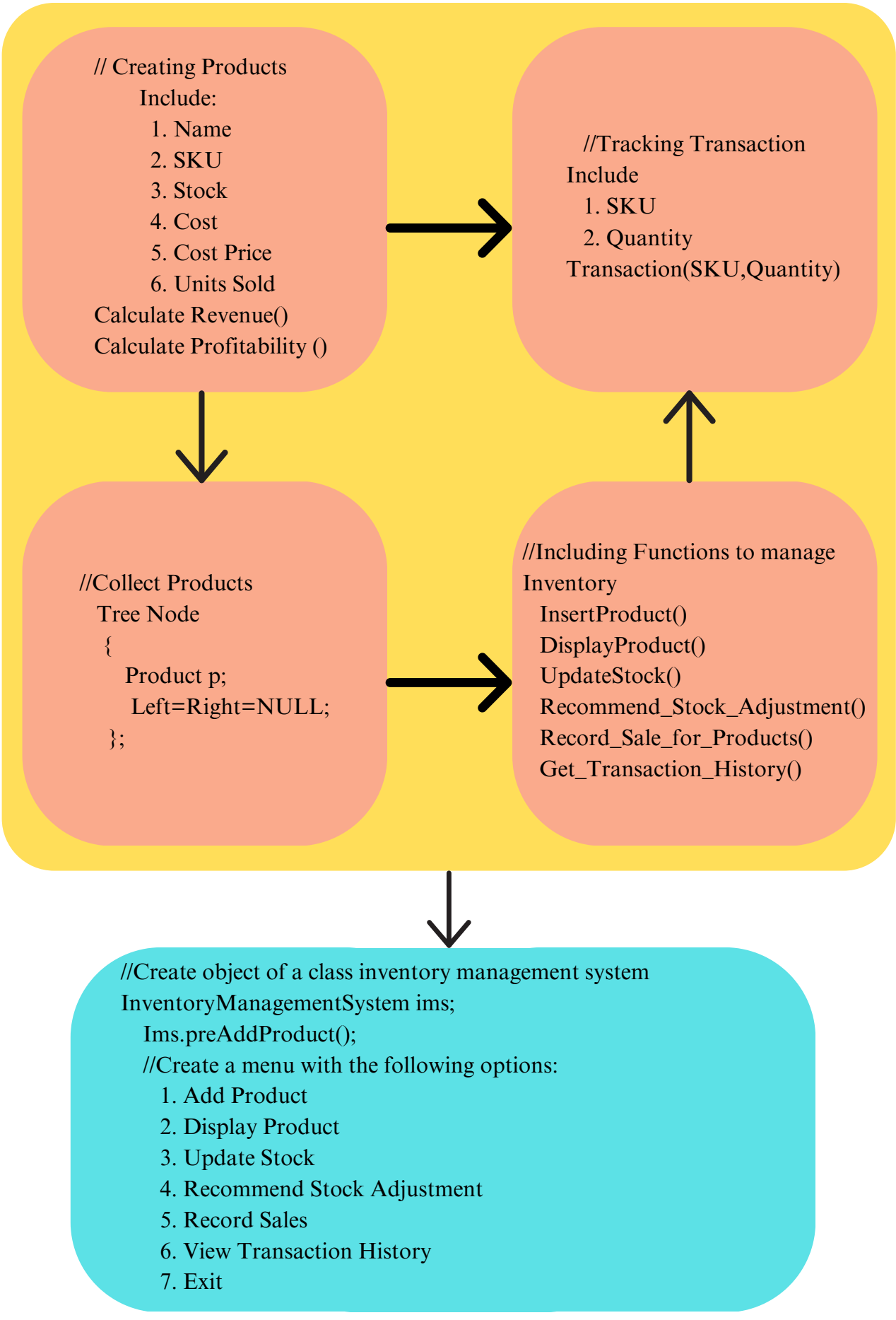
OVERVIEW OF INVENTORY MANAGEMENT SYSTEM

We have tried to implement an inventory management system using a binary search tree (BST) to store product information. It provides various functionalities, including adding products, displaying products, updating stock, recommending stock adjustments, recording sales, and viewing transaction history.

Main Functionality:

- Pre-adding Products: Initializes the inventory with a few pre-defined products.
- User Interaction Loop: Presents a menu with options to add products, display products, update stock, recommend stock adjustments, record sales, view transaction history, or exit.
- Product Addition: Prompts for product details and adds the product to the BST.
- Product Display: Displays product information (name, SKU, stock, cost, cost price, units sold, revenue, profitability) sorted by SKU.
- Stock Update: Prompts for SKU and quantity, updates stock in the BST, and handles underflow and stock alerts.
- Stock Adjustment Recommendations: Traverses the BST and recommends stock adjustments based on revenue and sales.
- Sales Recording: Traverses the BST, prompts for units sold for each product, updates stock and units sold, and handles stock exceedance.
- Transaction History View: Displays the transaction history (SKU and quantity) for all stock updates.

BLOCK DIAGRAM



KEY COMPONENTS

- Product: Represents a product with attributes like name, SKU, stock, cost, cost price, and units sold. The creation of new products takes place through this class.

- Transaction: Represents a stock update transaction with SKU and quantity. This part is responsible for maintaining transaction records. This includes the addition of stocks, sales of stocks, updation of stocks, etc.

- Tree Node: Represents a node in the BST, containing a Product object and pointers to left and right child nodes. We have used this data structure for maintaining records of products , for traversal and displaying of the stock information.

- Inventory Management System: Manages the inventory, including adding products, updating stock, recording sales, and displaying various information. The aim of this part is to manage the inventory system.

- main(): This is the driver part of the entire program. We can control the flow of the events, which would be executed using this part.

CODE

```
#include <iostream>
#include <vector>
#include <algorithm>
class Product
{
    public:
    std::string name;
    int sku;
    int stock;
    double cost; // Cost per unit
    double costPrice; // Cost price of the product
    int unitsSold; // Number of units sold

    Product(std::string n, int s, int st, double c, double cp)
        : name(n), sku(s), stock(st), cost(c), costPrice(cp), unitsSold(0) {}

   double calculateProfitability() const
   {
       // Adjusted profitability considering sales, remaining stock, and cost price
return unitsSold * (costPrice - cost) + (stock * cost);
    }
double calculateRevenue() const { return unitsSold * costPrice;
}
};
class Transaction
{
   public:
   int sku;
   int quantity;
   Transaction(int s, int q) : sku(s), quantity(q) {}
 };
class TreeNode
{
   public:
   Product product;
   TreeNode* left;
   TreeNode* right;
   TreeNode(const Product& p) : product(p), left(nullptr), right(nullptr) {}
};
```

```
class InventoryManagementSystem
{
   private:
   TreeNode* root;
    std::vector<Transaction> transactionHistory; // Transaction log
   // Helper function to insert a product into the BST
   TreeNode* insertProduct(TreeNode* node, const Product& product)
   {
       if (node == nullptr)
       {
          return new TreeNode(product);
       }
       if (product.sku < node->product.sku)
       {
          node->left = insertProduct(node->left, product);
       }
       else if (product.sku > node->product.sku)
       {
           node->right = insertProduct(node->right, product);
       }
       return node;
   }
// Helper function to display products in sorted order (in-order traversal)
void displayProducts(TreeNode* node) const
{
    if (node != nullptr) { displayProducts(node->left);
    std::cout << "Name: " << node->product.name
    << "\tSKU: " << node->product.sku
    << "\tStock: " << node->product.stock
    << "\tCost Price: $" << node->product.costPrice
    << "\tCost: $" << node->product.cost
    << "\tUnits Sold: " << node->product.unitsSold
    << "\tRevenue: $" << node->product.calculateRevenue()
    << "\tProfitability: $" << node->product.calculateProfitability()
    << "\n";
    displayProducts(node->right); }
}
```

```
// Helper function to update stock in the BST
 void updateStock(TreeNode* node, int sku, int quantity) {
if (node != nullptr) {
 if (sku == node->product.sku) {
// Update stock for the found product
 // Check for stock underflow
 if (quantity < 0 && node->product.stock + quantity < 0) {
std::cout << "Error: Stock underflow. Cannot decrease stock below 0.\n"; } else {
// Update stock
 node->product.stock += quantity; // Add stock for all quantities
transactionHistory.emplace_back(sku, quantity); // Log the transaction
// Check if stock is below 50 and display an alert if (node->product.stock < 50) {
std::cout << "Alert: Stock for " << node->product.name
 << " is below 50. Current stock: " << node->product.stock << "\n";
}
std::cout << "Stock updated successfully.\n"; }
} else if (sku < node->product.sku) {
 // Search in the left subtree updateStock(node->left, sku, quantity);
} else {
 // Search in the right subtree updateStock(node->right, sku, quantity);
}
 } else {
// Product with the specified SKU not found
std::cout << "Product with SKU " << sku << " not found.\n"; }
}
// Helper function to recommend stock adjustments based on revenue and sales void
recommendStockAdjustments(TreeNode* node) const {
if (node != nullptr) { recommendStockAdjustments(node->left);
double revenue = node->product.calculateRevenue(); int unitsSold = node-
>product.unitsSold;
if (revenue > 1000) {
 std::cout << "Recommend stocking more of " << node->product.name
<< ". Revenue: $" << revenue << "\n";
} else if (revenue < 4500) {
 std::cout << "Recommend stocking less of " << node->product.name
<< ". Revenue: $" << revenue << "\n"; }
if (unitsSold > 50) {
std::cout << "Product " << node->product.name << " is selling well. Consider promoting
it.\n";
} else if (unitsSold < 10) {
std::cout << "Product " << node->product.name << " has lower sales. Evaluate marketing
strategies.\n";
}
recommendStockAdjustments(node->right); }
}
```

```
public:
 InventoryManagementSystem() : root(nullptr) {}
// Function to pre-add some products void preAddProducts()
{
// Add pre-defined products to the inventory
addProduct(Product("Laptop", 1001, 50, 800.0, 700.0));
addProduct(Product("Smartphone", 1002, 30, 400.0, 350.0));
addProduct(Product("Tablet", 1003, 20, 300.0, 250.0));
}
void addProduct(const Product& product) { root =
insertProduct(root, product);
}
 void displayProducts() const {
std::cout << "Product List (Sorted by SKU):\n";
displayProducts(root);
 std::cout << "-----------------\n";
}
void updateStock(int sku, int quantity) {
 // Start the stock update from the root of the BST
updateStock(root, sku, quantity);
}
void recommendStockAdjustments() const { std::cout <<
"Stock Recommendations:\n";
recommendStockAdjustments(root); std::cout << "----------
----\n";
}
std::cout << "Product List (Sorted by SKU):\n";
displayProducts(root);
 std::cout << "-----------------\n";
}
void updateStock(int sku, int quantity) {
 // Start the stock update from the root of the BST
updateStock(root, sku, quantity);
}
void recommendStockAdjustments() const { std::cout <<
"Stock Recommendations:\n";
recommendStockAdjustments(root); std::cout << "----------
----\n";
}
private:
 // Helper function to record sales for each product (in-order
traversal)
```

```
void recordSalesForProduct(TreeNode* node)
{ if (node != nullptr) {
recordSalesForProduct(node->left);
int unitsSold;
 std::cout << "Enter units sold for product " <<
node->product.name
<< " (SKU: " << node->product.sku << "): ";
std::cin >> unitsSold;
// Check if units sold exceed available stock and
display an alert if (unitsSold > node-
>product.stock) {
std::cout << "Error: Units sold cannot exceed
available stock.\n"; } else {
// Update units sold for the product node-
>product.unitsSold += unitsSold;
updateStock(node, node->product.sku, -
unitsSold);
}
recordSalesForProduct(node->right); }
}
};
int main()
{ InventoryManagementSystem ims;
// Pre-add some products to the inventory
ims.preAddProducts();
while (true) {
 std::cout << "1. Add Product\n"; std::cout <<
"2. Display Products\n";
std::cout << "3. Update Stock\n";
 std::cout << "4. Recommend Stock
Adjustments\n"; std::cout << "5. Record
Sales\n";
 std::cout << "6. View Transaction History\n";
std::cout << "7. Exit\n";
 std::cout << "Enter your choice: ";
int choice;
 std::cin >> choice;
switch (choice) {
```

```
case 1: {
std::string name;
 int sku, stock;
 double cost, costPrice;
std::cout << "Enter product name: ";
std::cin.ignore(); std::getline(std::cin, name);
std::cout << "Enter SKU: "; std::cin >> sku;
std::cout << "Enter initial stock: "; std::cin >>
stock;
std::cout << "Enter cost per unit: $"; std::cin >>
cost;
std::cout << "Enter cost price: $"; std::cin >>
costPrice;
ms.addProduct(Product(name, sku, stock, cost,
costPrice));
break; }
case 2: ims.displayProducts(); break;
case 3: {
 int sku, quantity;
std::cout << "Enter SKU for stock update: ";
std::cin >> sku;
std::cout << "Enter quantity for stock update: ";
std::cin >> quantity;
ims.updateStock(sku, quantity);
break; }
case 4: ims.recommendStockAdjustments(); break;
case 5: ims.recordSales(); break;
case 6: {
 const auto& history =
ims.getTransactionHistory(); std::cout <<
"Transaction History:\n";
 for (const auto& transaction : history) {
std::cout << "SKU: " << transaction.sku <<
"\tQuantity: " << transaction.quantity << "\n";
}
 std::cout << "-----------------\n";
break; }
case 7:
 std::cout << "Exiting program.\n"; return 0;
default:
 std::cout << "Invalid choice. Try again.\n";
} }
return 0;
}
```