REPUBLIC OF THE PHILIPPINES

BICOL UNIVERSITY

BICOL UNIVERSITY POLANGUI

MEMBERS:

Curt Justin N. Reodique.

Jayvee Franco

Mhelarry Valeza

SUBJECT: **DSA**

COURSE/YEAR: **BSIS - 2A**

PROF: **Ms. Khrstine Botin**

TITLE: WELCOME TO THE SHOPPING CART SYSTEM!

PROJECT OVERVIEW

- The Shopping Cart System is a simple C++ application that uses a binary search tree (BST) to organize and manage items in a shopping cart. Each item in the cart is represented as a node in the tree, where the nodes are ordered by their price. The user can add items to the cart, display them using different traversal methods (Pre-order, In-order, and Post-order), and exit the system. The program has a maximum limit of 5 items in the cart.

HOW TO RUN THE CODE

Environment Requirements:

- Any IDE or code editor that supports C++ (e.g., Code::Blocks, Visual Studio Code, or CLion).

Interacting with the Program:

a. **Choose Option 1** to add items to the cart by entering the item details (order ID, name, and price).
b. **Choose Option 2** to display the cart items in a specified traversal order (Pre-order, In-order, or Post-order).
c. **Choose Option 3** to exit the program.

DESCRIPTION OF EACH FUNCTIONALITY

1. Node Structure

- Represents a single item in the shopping cart.
- Contains:

➢ **orderId**: A unique identifier for the item.
➢ **itemName**: The name of the item.
➢ **Price**: The price of the item.
➢ **Left and right**: Pointers to the left and right child nodes in the BST.

2. ShoppingCart Class

- Manages the binary search tree and provides methods to interact with it.

Private Members:

- **Node* root:** Root node of the binary search tree.
- **Int itemCount:** Tracks the current number of items in the cart.
- **Const int MAX_ITEMS:** Sets the limit on the number of items.

Private Methods:

Node* insert(Node* node, int orderId, string itemName, double price):

- Recursively inserts a new item (node) into the tree based on the price.
- Items with lower prices go to the left subtree, and higher prices go to the right.

Void preOrderTraversal(Node* node, string& result):

- Visits the root, left subtree, and right subtree in order.
- Appends each item's price to the result string.

Void inOrderTraversal(Node* node, string& result):

- Visits the left subtree, root, and right subtree in order.
- Appends each item's price to the result string.

Void postOrderTraversal(Node* node, string& result):

- Visits the left subtree, right subtree, and root in order.
- Appends each item's price to the result string.

Public Methods:

ShoppingCart(): Constructor that initializes the cart with an empty tree and sets the item count to 0.

Void addItem(int orderId, string itemName, double price):

- Adds a new item to the cart.
- Checks if the cart has reached its maximum capacity before adding the item.
- Outputs a success message with the item's name and price.

Void displayItems(string traversalType):

- Displays the cart items in the specified traversal order.
- Supports three traversal types: Pre-order, In-order, and Post-order.
- Formats the output prices with "Php" and two decimal points.
- Removes the trailing comma from the output.

3. Main Function

- Manages the user interface and interactions with the shopping cart system.
- Features:
- Displays a menu with three options:

1. Add an item to the cart.
2. Display the cart items in a specified traversal order.
3. Exit the program.

- Prompts the user to enter item details (ID, name, and price) when adding items.
- Allows the user to choose the traversal type for displaying items.

- Exits gracefully when the user chooses to quit.

4. **Search for an Item (Searching)**

- Function: searchItem()
- Process:
  - The function searches the BST for a node with the specified orderId.
  - It recursively compares the orderId with the current node:
  - If orderId is smaller, search the left subtree.
  - If orderId is larger, search the right subtree.
  - If a match is found, the item's details (orderId, itemName, and price) are displayed.
  - If no match is found, a message indicating that the item is not in the cart is displayed.

5. **Delete an Item (Deletion)**

- Function: deleteItem()
- Process:
  - Removes an item with a specified orderId from the BST.

Three Cases for Deletion:

1. Node has no children (leaf node):
   - The node is deleted, and its parent's pointer is set to nullptr.
2. Node has one child:
   - The node is deleted, and its parent is linked directly to the child.
3. Node has two children:
   - The node is replaced with its in-order successor (smallest node in its right subtree).
   - The in-order successor is then deleted from the right subtree.

- After deletion, the itemCount is decremented.

EXAMPLE PROGRAM FLOW

1. Welcome Message:

- Welcome to the Shopping Cart System!

2. Add Items:

- Enter order ID: 101
- Enter item name: Laptop
- Enter item price: 80000.00
- Item 'Laptop' added to cart for Php 80000.00

3. Display Items (In-order):

- Enter traversal type (Pre-order, In-order, Post-order):
- In-order
- In-order: Php 60000.00, Php 80000.00

4. Exit the Program:

- Exiting the system. Goodbye!

DETAILED STEPS

Option 1: Add an Item

4. The user is prompted to enter:
   - orderId: A unique number for the item.
   - itemName: A descriptive name for the item.
   - Price: The cost of the item.

5. The program checks if the cart is full (maximum of 5 items):
    - If the cart is full, a message is displayed, and the user is redirected to the menu.
    - Otherwise, the item is added to the cart, and a success message is displayed.
6. The item is inserted into the binary search tree, sorted by its price.

Option 2: Display Items

1. The user is prompted to specify a traversal type:
    - Pre-order: Displays items in root-left-right order.
    - In-order: Displays items in left-root-right order (sorted by price).
    - Post-order: Displays items in left-right-root order.

2. The corresponding traversal function is called, and the cart's prices are added to a formatted string (ex: "Php 60000.00, Php 80000.00").

3. The final output displays the traversal type followed by the prices:
    - In-order: Php 60000.00, Php 80000.00

Option 3: Exit the System

1. The program outputs a goodbye message and terminates:
- Exiting the system. Goodbye!

2. All memory and resources are released as the program ends.

TRAVERSALS AND DISPLAY

1. The user selects a traversal type.
2. The program traverses the binary search tree based on the chosen order:
    - **Pre-order:** Visits root, then left subtree, then right subtree.
    - **In-order:** Visits left subtree, root, then right subtree.
    - **Post-order:** Visits left subtree, right subtree, then root.
3. After the traversal is complete:
    - The formatted list of prices is displayed.
    - The traversal type is printed at the end for clarity:
    - Pre-order: Php 80000.00, Php 60000.00
    - Traversal: Pre-order

PROJECT SUMMARY

- The Shopping Cart System is a console-based application that simulates a shopping cart using a Binary Search Tree (BST). Each item added to the cart is stored as a node in the BST, organized by the item's price. The system allows users to manage their cart by adding items and viewing them in different traversal orders. The application ensures that the cart has a maximum capacity of 5 items and provides a structured way to interact with the data through pre-order, in-order, and post-order traversals.
  Summary of Functionality:
- This project blends practical functionality with the learning experience of working with data structures (BST) to create a realistic and interactive shopping cart system.

KEY FEATURES

1. Add Items to Cart:
- Users can add items with details such as an order ID, item name, and price.
- Items are stored in a BST, organized by price.
2. Display Items:
- Users can view items in the cart in three traversal formats:

a. **Pre-order:** Displays items in root-left-right order.
b. **In-order:** Displays items sorted by price (left-root-right).
c. **Post-order:** Displays items in left-right-root order.

- The formatted output displays prices prefixed with “Php” and lists the selected traversal type at the end.

3. Cart Capacity Management:
- Ensures a maximum of 5 items can be added.
- Displays an error message if the user tries to exceed the limit.
4. User-Friendly Interface:
- Intuitive menu system for navigating options.
- Clear messages guide users through each step of the process.

USE CASES

1. Shopping Simulation:
   - This application mimics a real-world shopping cart where items are added and displayed based on the user’s preference.
2. Learning Tool:
   - The project demonstrates the use of BST in managing hierarchical data and implementing traversals.
3. Data Organization:
   - Organizes items efficiently by price, allowing structured access and display of data.

CODE:

```
#include <iostream>

#include <iomanip>

#include <string>

Using namespace std;


// Node structure for the Binary Search Tree

Struct Node {

    Int orderId;

    String itemName;

    Double price;

    Node* left;

    Node* right;


    Node(int id, string name, double price) : orderId(id), itemName(name), price(price), left(nullptr), right(nullptr) {}

};
```

```
// Binary Search Tree class
Class ShoppingCart {
Private:
  Node* root;
  Int itemCount; // Track the number of items in the cart
  Const int MAX_ITEMS = 5; // Maximum allowed items

  // Helper function to insert an item
  Node* insert(Node* node, int orderId, string itemName, double price) {
    If (node == nullptr) {
      Return new Node(orderId, itemName, price);
    }
    If (price < node->price) {
      Node->left = insert(node->left, orderId, itemName, price);
    } else {
      Node->right = insert(node->right, orderId, itemName, price);
    }
    Return node;
  }

  // Helper function for searching an item
  Node* search(Node* node, int orderId) {
    If (node == nullptr || node->orderId == orderId) {
      Return node;
    }
    If (orderId < node->orderId) {
      Return search(node->left, orderId);
    }
    Return search(node->right, orderId);
  }

  // Helper function to find the minimum value node
  Node* findMin(Node* node) {
    While (node->left != nullptr) {
      Node = node->left;
    }
    Return node;
```

```
    }

    // Helper function to delete a node
    Node* deleteNode(Node* node, int orderId) {
        If (node == nullptr) {
            Return node;
        }
        If (orderId < node->orderId) {
            Node->left = deleteNode(node->left, orderId);
        } else if (orderId > node->orderId) {
            Node->right = deleteNode(node->right, orderId);
        } else {
            // Node with only one child or no child
            If (node->left == nullptr) {
                Node* temp = node->right;
                Delete node;
                Return temp;
            } else if (node->right == nullptr) {
                Node* temp = node->left;
                Delete node;
                Return temp;
            }

            // Node with two children: Get the in-order successor (smallest in the right subtree)
            Node* temp = findMin(node->right);
            Node->orderId = temp->orderId;
            Node->itemName = temp->itemName;
            Node->price = temp->price;

            // Delete the in-order successor
            Node->right = deleteNode(node->right, temp->orderId);
        }
        Return node;
    }

    // Helper function for pre-order traversal
    Void preOrderTraversal(Node* node, string& result) {
```

```
        If (node == nullptr) return;
        Result += “Php “ + to_string(node->price) + “, “;
        preOrderTraversal(node->left, result);
        preOrderTraversal(node->right, result);
    }

    // Helper function for in-order traversal
    Void inOrderTraversal(Node* node, string& result) {
        If (node == nullptr) return;
        inOrderTraversal(node->left, result);
        result += “Php “ + to_string(node->price) + “, “;
        inOrderTraversal(node->right, result);
    }

    // Helper function for post-order traversal
    Void postOrderTraversal(Node* node, string& result) {
        If (node == nullptr) return;
        postOrderTraversal(node->left, result);
        postOrderTraversal(node->right, result);
        result += “Php “ + to_string(node->price) + “, “;
    }

Public:
    ShoppingCart() : root(nullptr), itemCount(0) {}

    // Public method to add an item to the cart
    Void addItem(int orderId, string itemName, double price) {
        If (itemCount >= MAX_ITEMS) {
            Cout << “The cart is full! You can only have up to “ << MAX_ITEMS << “ items.” << endl;
            Return;
        }
        Root = insert(root, orderId, itemName, price);
        itemCount++;
        cout << “Item ‘” << itemName << “’ added to cart for Php “ << fixed << setprecision(2) << price << endl;
    }

    // Public method to search for an item
```

```
  Void searchItem(int orderId) {
      Node* found = search(root, orderId);
      If (found != nullptr) {
          Cout << “Item found: “ << endl;
          Cout << “Order ID: “ << found->orderId << endl;
          Cout << “Item Name: “ << found->itemName << endl;
          Cout << “Price: Php “ << fixed << setprecision(2) << found->price << endl;
      } else {
          Cout << “Item with Order ID “ << orderId << “ not found.” << endl;
      }
  }

  // Public method to delete an item
  Void deleteItem(int orderId) {
      If (root == nullptr) {
          Cout << “The cart is empty!” << endl;
          Return;
      }
      Root = deleteNode(root, orderId);
      Cout << “Item with Order ID “ << orderId << “ has been deleted from the cart.” << endl;
      itemCount--; // Decrease the item count
  }

  // Public method to display items in a specified traversal order
  Void displayItems(string traversalType) {
      If (root == nullptr) {
          Cout << “The cart is empty!” << endl;
          Return;
      }
      String result;
      If (traversalType == “Pre-order”) {
          preOrderTraversal(root, result);
      } else if (traversalType == “In-order”) {
          inOrderTraversal(root, result);
      } else if (traversalType == “Post-order”) {
          postOrderTraversal(root, result);
      } else {
```

```
        Cout << “Invalid traversal type!” << endl;
        Return;
    }
    // Remove the trailing “, “ and display the result
    If (!result.empty()) result.pop_back(), result.pop_back();
    Cout << traversalType << “: “ << result << endl;
  }
};

Int main() {
  ShoppingCart cart;
  Int choice, orderId;
  String itemName, traversalType;
  Double price;

  Cout << “Welcome to the Shopping Cart System!” << endl;

  While (true) {
    Cout << “\nMenu:” << endl;
    Cout << “1. Add an item to the cart” << endl;
    Cout << “2. Display cart items (Pre-order, In-order, Post-order)” << endl;
    Cout << “3. Search for an item” << endl;
    Cout << “4. Delete an item from the cart” << endl;
    Cout << “5. Exit” << endl;
    Cout << “\nEnter your choice: “;
    Cin >> choice;

    Switch (choice) {
      Case 1:
        Cout << “Enter order ID: “;
        Cin >> orderId;
        Cout << “Enter item name: “;
        Cin.ignore();
        Getline(cin, itemName);
        Cout << “Enter item price: “;
        Cin >> price;
        Cart.addItem(orderId, itemName, price);
        Break;
```

```
        Case 2:
            Cout << “Enter traversal type (Pre-order, In-order, Post-order): “ << endl;
            Cin.ignore();
            Getline(cin, traversalType);
            Cart.displayItems(traversalType);
            Break;
        Case 3:
            Cout << “Enter the Order ID to search: “;
            Cin >> orderId;
            Cart.searchItem(orderId);
            Break;
        Case 4:
            Cout << “Enter the Order ID to delete: “;
            Cin >> orderId;
            Cart.deleteItem(orderId);
            Break;
        Case 5:
            Cout << “Exiting the system. Goodbye!” << endl;
            Return 0;
        Default:
            Cout << “Invalid choice. Please try again.” << endl;
    }
  }
  Return 0;
}
```