Includes and Namespace

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
```

- `#include <iostream>`: Includes the standard library for input/output operations.
- `#include <vector>`: Provides support for the `std::vector` container, used for dynamic arrays.
- `#include <queue>`: Includes support for queue operations, though it isn't directly used in this code.
- `using namespace std`: Allows usage of standard library classes and functions without the `std::` prefix.

Binary Tree Node Structure

```
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};
```

- `struct Node`: Defines a structure to represent a binary tree node.
- `int data`: Holds the value of the node.
- `Node* left`: Pointer to the left child.
- `Node* right`: Pointer to the right child.
- Constructor `Node(int val)`: Initializes the `data` field with `val` and sets child pointers to `nullptr`.

Function Prototypes

```
void binaryTreeMenu();
void binarySearchTreeMenu();
void heapMenu();
```

Declares menu functions for different data structures:

- `binaryTreeMenu()`: Handles binary tree operations.
- `binarySearchTreeMenu()`: Handles binary search tree (BST) operations.

- `heapMenu()`: Handles heap operations.

Main Menu

```
int main() {
    int choice;
    while (true) {
        cout << "\n==== Interactive Data Structures Menu ====\n";
        cout << "1. Binary Tree Operations\n";
        cout << "2. Binary Search Tree (BST) Operations\n";
        cout << "3. Heap Operations\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: binaryTreeMenu(); break;
            case 2: binarySearchTreeMenu(); break;
            case 3: heapMenu(); break;
            case 4: exit(0);
            default: cout << "Invalid choice. Please try again.\n";
        }
    }
    return 0;
}
```

- **Functionality**: Provides an interactive menu for the user to select the desired operation.
- `while (true)`: Creates an infinite loop for the menu system.
- `cout` and `cin`: Display menu options and take user input.
- `switch (choice)`: Executes the corresponding function based on user input.
    - Case 1: Calls `binaryTreeMenu`.
    - Case 2: Calls `binarySearchTreeMenu`.
    - Case 3: Calls `heapMenu`.
    - Case 4: Exits the program using `exit(0)`.

Binary Tree Menu

```
void binaryTreeMenu() { ... }
```

- Handles operations like insertion, traversal, searching, and deletion in a binary tree.
- `root`: Pointer to the root of the binary tree.
- `while (true)`: Displays the binary tree operations menu until the user chooses to return to the main menu.

Binary Tree Operations

Insert into Binary Tree

```
Node* insertBinaryTree(Node* root, int value) {
    if (!root) return new Node(value);
    if (!root->left)
        root->left = insertBinaryTree(root->left, value);
    else
        root->right = insertBinaryTree(root->right, value);
    return root;
}
```

- Recursively inserts a new node into the first available position in level order.
- If `root` is `nullptr`, a new node is created and returned.
- If the left child is `nullptr`, inserts the value into the left subtree.
- Otherwise, inserts into the right subtree.

Traversals

- **Preorder**: Visits `root`, `left`, then `right`.
- **Inorder**: Visits `left`, `root`, then `right`.
- **Postorder**: Visits `left`, `right`, then `root`.

Search Binary Tree

```
bool searchBinaryTree(Node* root, int value) {
    if (!root) return false;
    if (root->data == value) return true;
    return searchBinaryTree(root->left, value) || searchBinaryTree(root-
>right, value);
}
```

- Recursively searches the tree for a value.
- Returns `true` if found; otherwise, continues searching.

Delete Binary Tree

```
Node* deleteBinaryTree(Node* root, int key) { ... }
```

- Deletes a node with the given value.
- If the node is found and has no children, deletes the node.
- Recursively adjusts left and right children as needed.

Binary Search Tree (BST) Menu and Operations

Menu

- Similar to the binary tree menu, but operations are optimized for BST rules.

Insert BST

```
Node* insertBST(Node* root, int value) {
    if (!root) return new Node(value);
    if (value < root->data)
        root->left = insertBST(root->left, value);
    else
        root->right = insertBST(root->right, value);
    return root;
}
```

- Inserts nodes following BST properties: smaller values go left, larger go right.

Search BST

```
bool searchBST(Node* root, int value) { ... }
```

- Searches for a value in a BST using binary search logic.

Delete BST

```
Node* deleteBST(Node* root, int value) { ... }
```

- Handles all cases for deletion:
    - No child.
    - One child.
    - Two children: Finds the in-order successor (smallest value in the right subtree).

Heap Menu and Operations

Menu

- Displays heap options (e.g., insert, build max-heap, print heap).

Insert Heap

```
void insertHeap(vector<int>& heap, int value) { ... }
```

- Inserts a value and reorders the heap to maintain the max-heap property.

Heapify

```
void heapify(vector<int>& heap, int n, int i) { ... }
```

- Recursively ensures the heap property by comparing the parent with its children.

Build Heaps

- **Max-Heap**: Reorders elements so the largest element is the root.
- **Min-Heap**: Reorders elements so the smallest element is the root.

Print Heap

```
void printHeap(vector<int>& heap) { ... }
```

- Prints all elements in the heap.
