Aim: Binary Search Tree Traversal.

Objectives: The objective is to understand and implement the traversal methods of a Binary Search Tree (BST), specifically InOrder, PreOrder, and PostOrder. These methods involve visiting the nodes of the tree in different sequences to explore the tree's data.

Tools Used: Visual Studio Code.

Concept:

The concept of InOrder, PreOrder, and PostOrder traversal in a Binary Search Tree (BST) refers to the systematic way of visiting all the nodes in the tree. In InOrder traversal, the nodes are visited in the order: Left subtree → Data → Right subtree. In PreOrder traversal, the sequence is Data → Left subtree → Right subtree, starting with the data first. In PostOrder traversal, the nodes are visited in the order: Left subtree → Right subtree → Data, ending with the data last. These methods help explore and process all the nodes in the tree systematically.

Problem Statement:

Implement Binary Search Tree Traversal

1) In-Order
2) Pre-Order
3) Post-Order

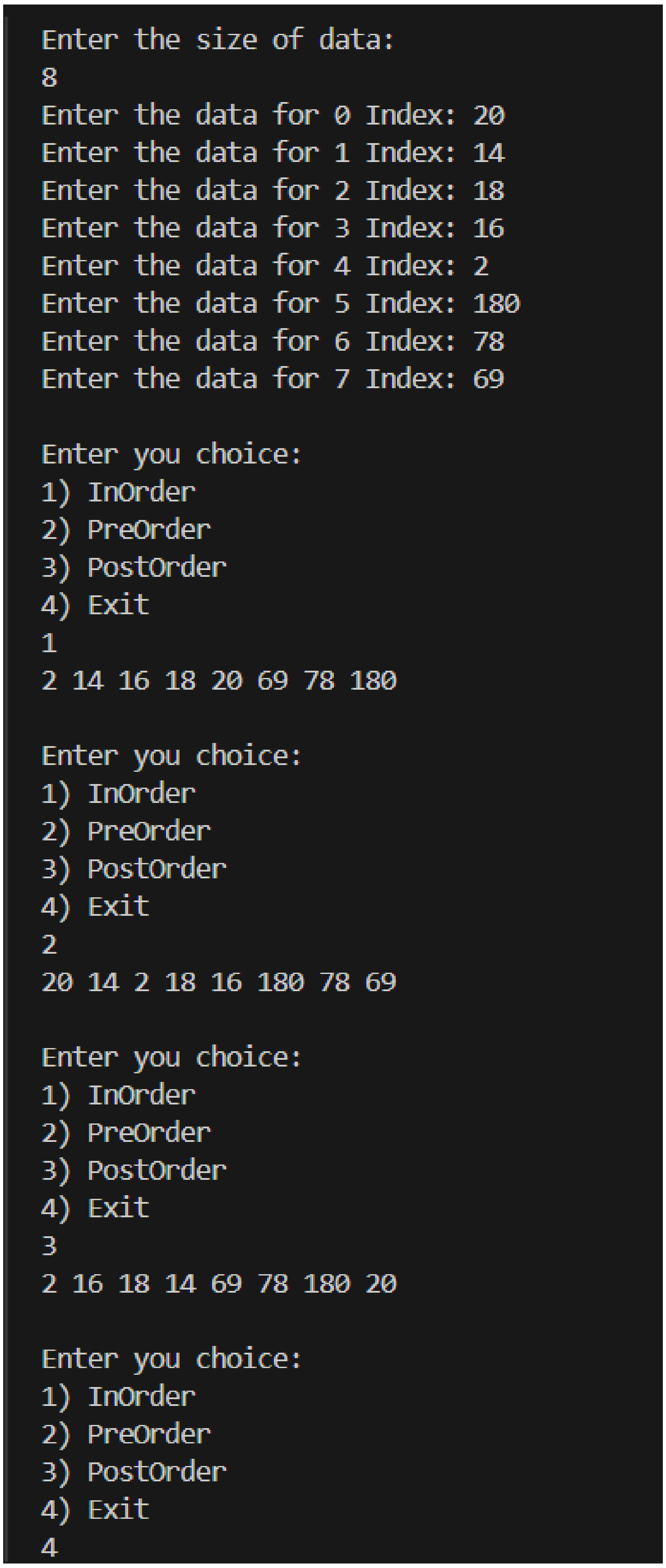
Solution:

```
#include <iostream>
using namespace std;
#define max 100

int n, treeData[max], ch;

struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
    *list = NULL,
    *p, *q, *r, *s, *temp, *root = NULL, *current, *parent, *lastVisited,
*peekNode, *stack[max];

class BST
{
public:
    void menu()
    {

        do
        {
```

```cpp
        cout << endl;
        cout << "Enter you choice:" << endl;
        cout << "1) InOrder" << endl;
        cout << "2) PreOrder" << endl;
        cout << "3) PostOrder" << endl;
        cout << "4) Exit" << endl;

        cin >> ch;

        switch (ch)
        {
        case 1:
            InOrder();
            break;
        case 2:
            PreOrder();
            break;
        case 3:
            PostOrder();
            break;

        default:
            cout << "Enter proper value." << endl;
            break;
        }
    } while (ch != 4);
}

void InOrder()
{
    if (root == NULL)
    {
        cout << "Tree is empty!" << endl;
        return;
    }

    int top = -1;
    current = root;

    while (current != NULL || top != -1)
    {
        while (current != NULL)
        {
            stack[++top] = current;
            current = current->prev;
        }

        current = stack[top--];
```

```cpp
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

    void PreOrder()
    {
        if (root == NULL)
        {
            cout << "Tree is empty!" << endl;
            return;
        }

        int top = -1;
        stack[++top] = root;

        while (top != -1)
        {
            current = stack[top--];
            cout << current->data << " ";

            if (current->next != NULL)
                stack[++top] = current->next;
            if (current->prev != NULL)
                stack[++top] = current->prev;
        }
        cout << endl;
    }

    void PostOrder()
    {
        if (root == NULL)
        {
            cout << "Tree is empty!" << endl;
            return;
        }

        int top = -1;
        lastVisited = NULL;
        current = root;

        while (current != NULL || top != -1)
        {
            if (current != NULL)
            {
                stack[++top] = current;
                current = current->prev;
            }
            else
            {
                peekNode = stack[top];

                if (peekNode->next != NULL && lastVisited != peekNode->next)
                {
                    current = peekNode->next;
                }
                else
                {
                    cout << peekNode->data << " ";
                    lastVisited = stack[top--];
                }
            }
        }
        cout << endl;
    }

    void input()
    {
        cout << "Enter the size of data:" << endl;
        cin >> n;

        treeData[n];
        for (int i = 0; i < n; i++)
        {
            cout << "Enter the data for " << i << " Index: ";
            cin >> treeData[i];
        }

        for (int i = 0; i < n; i++)
        {
            createTree(treeData[i]);
        }
    }

    void createTree(int key)
    {
        p = (struct node *)malloc(sizeof(node));

        p->data = key;
        p->prev = NULL;
        p->next = NULL;

        if (root == NULL)
        {
            root = p;
            return;
        }

        current = root;
        parent = NULL;

        while (current != NULL)
        {
            parent = current;
            if (key < current->data)
            {
                current = current->prev;
            }
            else
            {
                current = current->next;
            }
        }

        if (key < parent->data)
        {
            parent->prev = p;
        }
        else
        {
            parent->next = p;
        }
    }
};
int main()
{
    BST bst;
    bst.input();
    bst.menu();
    return 0;
}
```

Output:



```
Enter the size of data:
8
Enter the data for 0 Index: 20
Enter the data for 1 Index: 14
Enter the data for 2 Index: 18
Enter the data for 3 Index: 16
Enter the data for 4 Index: 2
Enter the data for 5 Index: 180
Enter the data for 6 Index: 78
Enter the data for 7 Index: 69

Enter you choice:
1) InOrder
2) PreOrder
3) PostOrder
4) Exit
1
2 14 16 18 20 69 78 180

Enter you choice:
1) InOrder
2) PreOrder
3) PostOrder
4) Exit
2
20 14 2 18 16 180 78 69

Enter you choice:
1) InOrder
2) PreOrder
3) PostOrder
4) Exit
3
2 16 18 14 69 78 180 20

Enter you choice:
1) InOrder
2) PreOrder
3) PostOrder
4) Exit
4
```

Observation: When performing traversal operations on a Binary Search Tree (BST), each type of traversal (InOrder, PreOrder, PostOrder) works differently. InOrder traversal visits nodes in the left-to-right order, which processes the nodes after visiting their left children and before their right children. PreOrder traversal starts at the root, visiting the data first, then moving to the left and right children. PostOrder traversal visits the left and right children before the data, ensuring that the children are processed first. In all cases, traversal involves following a specific sequence to visit and process all the nodes in the tree.