|  |  |
| --- | --- |
| Name | Taaha Hussain Khan |
| Roll No: | L1F21BSCS0917 |
| Section | D12 |

**Graded Lab**

**Task1**

#include <iostream>

using namespace std;

class BST {

int data;

BST\* left;

BST\* right;

public:

BST()

{

data = 0;

right = NULL;

left = NULL;

}

// constructor

BST(int d)

{

data = d;

right = NULL;

left = NULL;

}

BST\* Insert(BST\* root, int d)

{

if (root == NULL)

{

return new BST(d);

}

if (d > root->data)

{

root->right = Insert(root->right, d);

}

else if (d < root->data)

{

root->left = Insert(root->left, d);

}

return root;

}

void Inorder(BST\* root)

{

if (!root)

{

return;

}

Inorder(root->left);

cout << root->data << ", ";

Inorder(root->right);

}

void Preorder(BST\* root)

{

if (!root)

{

return;

}

cout << root->data << ", ";

Preorder(root->left);

Preorder(root->right);

}

void Postorder(BST\* root)

{

if (!root)

{

return;

}

Postorder(root->left);

Postorder(root->right);

cout << root->data << ", ";

}

bool search(BST\* root, int key)

{

if (root == NULL)

{

return false;

}

if (root->data == key)

{

return true;

}

if (key > root->data)

{

return search(root->right, key);

}

else

{

return search(root->left, key);

}

return false;

}

};

int main()

{

BST b, \*root = NULL;

root = b.Insert(root, 50);

b.Insert(root, 30);

b.Insert(root, 20);

b.Insert(root, 40);

b.Insert(root, 70);

b.Insert(root, 60);

b.Insert(root, 80);

cout << "\n\nIn order traversal: \n";

b.Inorder(root);

cout << "\n\nPre order traversal: \n";

b.Preorder(root);

cout << "\n\nPost order traversal: \n";

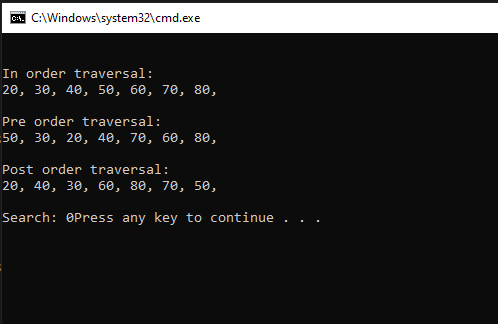
b.Postorder(root);

cout << "\n\nSearch: ";

cout << b.search(root, 400);

return 0;

}



**Task2**

#include<iostream>

using namespace std;

class Node{

public:

int data;

Node\* nextPtr;

Node\* prevPtr;

Node(int data){

this->data = data;

nextPtr = NULL;

prevPtr = NULL;

}

};

void insertAtHead(Node\* &head, int data){

Node\* n = new Node(data);

n->nextPtr = head;

if (head != NULL){

head->prevPtr = n;

}

head = n;

}

void insertAtTail(Node\* &head, int data){

if (head == NULL){

insertAtHead(head, data);

return;

}

Node\* n = new Node(data);

Node\* temp = head;

while (temp->nextPtr != NULL){

temp = temp->nextPtr;

}

temp->nextPtr = n;

n->prevPtr = temp;

}

void display(Node\* head){

Node\* temp = head;

cout << "\n\tNULL -> ";

while (temp != NULL){

cout << temp->data << " -> ";

temp = temp->nextPtr;

}

cout << "NULL\n" << endl;

}

void reverseList(Node\* &head) {

if (head != NULL) {

Node\* prevNode = head;

Node\* temp = head;

Node\* curNode = head->nextPtr;

prevNode->nextPtr = NULL;

prevNode->prevPtr = NULL;

while (curNode != NULL) {

temp = curNode->nextPtr;

curNode->nextPtr = prevNode;

prevNode->prevPtr = curNode;

prevNode = curNode;

curNode = temp;

}

head = prevNode;

}

}

int main(){

Node\* head = NULL;

insertAtTail(head, 1);

insertAtTail(head, 2);

insertAtTail(head, 3);

insertAtTail(head, 4);

display(head);

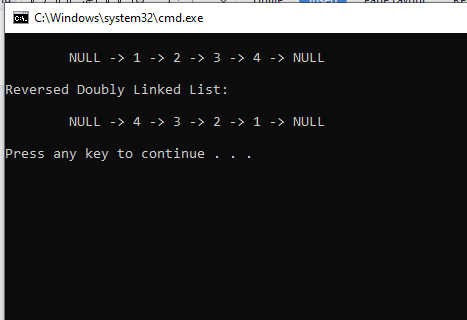
cout << "Reversed Doubly Linked List: " << endl;

reverseList(head);

display(head);

return 0;

}



**Task3**

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node \*prev;

Node(int data)

{

this->data = data;

next = NULL;

prev = NULL;

}

};

void inserAtHead(Node\* &head, int data)

{

Node\* newNode = new Node(data);

newNode->data = data;

newNode->prev = NULL;

newNode->next = head;

if (head != NULL)

{

head->prev = newNode;

}

head = newNode;

}

void sort(Node\* head)

{

Node\* temp1 = head;

while (temp1 != NULL)

{

Node\* temp2 = temp1 -> next;

while (temp2 != NULL)

{

if (temp1->data > temp2->data)

{

int temp = temp1->data;

temp1->data = temp2->data;

temp2->data = temp;

}

temp2 = temp2->next;

}

temp1 = temp1->next;

}

}

void DeleteNode(Node\* &head, Node\* todelete)

{

if (head == NULL || todelete == NULL)

{

return;

}

if (head == todelete)

{

head = todelete->next;

}

if (todelete->next != NULL)

{

todelete->next->prev = todelete->prev;

}

if (todelete->prev != NULL)

{

todelete->prev->next = todelete->next;

}

free(todelete);

}

void removeDuplicates(Node\* &head)

{

if (head == NULL || head->next == NULL)

{

return;

}

Node\* temp1 = head;

Node\* temp2 = head;

for (temp1 = head; temp1 != NULL; temp1 = temp1->next)

{

temp2 = temp1->next;

while (temp2 != NULL)

{

if (temp1->data == temp2->data)

{

Node\* curr = temp2->next;

DeleteNode(head, temp2);

temp2 = curr;

}

else

{

temp2 = temp2->next;

}

}

}

}

void display(Node\* head){

Node\* temp = head;

cout << "\n\tNULL -> ";

while (temp != NULL){

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n" << endl;

}

int main()

{

Node\* head = NULL;

cout << "Original Linked List: " << endl ;

inserAtHead(head, 10);

inserAtHead(head, 13);

inserAtHead(head, 11);

inserAtHead(head, 10);

inserAtHead(head, 13);

inserAtHead(head, 18);

inserAtHead(head, 10);

display(head);

sort(head);

cout << "Sorted Linked List: " << endl;

display(head);

removeDuplicates(head);

cout << "After removing duplicate values: " << endl;

display(head);

return 0;

}

