Assignment 2

Dated Dec 2nd, 2024

Problem Statement

Program in C to perform the following operations with binary search trees (BST):

1. Creating a BST
2. Inserting elements
3. Pre-order traversal
4. In-order traversal
5. Post-order traversal
6. Deleting elements
7. Displaying raw nodes

Algorithm

Input

Function(s) used for taking input: bst_init

Output

Methods used for displaying: bst_pre_order, bst_in_order, bst_post_order, bst_array_view.

Data structure used

We used a binary search tree data structure here with a pointer to the left and right and each pointer containing an integer value.

Step 1: Start.

Step 2: Define the structure BST with the following members: int val for the node value, struct BST* lhs as the pointer to the left child, and struct BST* rhs as the pointer to the right child.

Binary Search Tree Initialization (bst_init)

Step 3: Prompt the user to input the root value or -1 to quit.

Step 4: Read the value val. If input is invalid, terminate with an error.

Step 5: If val is -1, return NULL as the tree is empty and go to Step 35.
Step 6: Allocate memory for the root node. If allocation fails, terminate with an error.
Step 7: Assign val to the root node, set lhs and rhs to NULL.
Step 8: Start a loop to accept additional elements until the user enters -1.
Step 9: Prompt the user to input data or -1 to quit.
Step 10: Read the value val. If input is invalid, terminate with an error.
Step 11: If val is -1, break the loop and go to Step 13.
Step 12: Call _bst_insert_elem with the root and val. Go to Step 9.
[End of loop defined at Step 8].
Step 13: Return the root node. [End of function bst_init].

Insert Element (_bst_insert_elem)

Step 14: Check if the current node (bst) is NULL.
Step 15: If bst is NULL, allocate memory for a new node. If allocation fails, terminate with an error.
Step 16: Assign elem to the new node's val, and set its lhs and rhs to NULL.
Step 17: Return the newly created node and go to Step 35.
Step 18: If elem is less than bst->val, recursively call _bst_insert_elem on the left subtree. Go to Step 20.
Step 19: If elem is greater than bst->val, recursively call _bst_insert_elem on the right subtree.
Step 20: Return the modified bst node. [End of function _bst_insert_elem].

Traversals

Pre-order Traversal (bst_pre_order)
Step 21: If the node is NULL, return.
Step 22: Print the value of the current node.
Step 23: Recursively call bst_pre_order on the left subtree.
Step 24: Recursively call bst_pre_order on the right subtree.
[End of function bst_pre_order].

In-order Traversal (bst_in_order)
Step 25: If the node is NULL, return.
Step 26: Recursively call bst_in_order on the left subtree.
Step 27: Print the value of the current node.
Step 28: Recursively call bst_in_order on the right subtree.
[End of function bst_in_order].

Post-order Traversal (bst_post_order)

Step 29: If the node is NULL, return.

Step 30: Recursively call bst_post_order on the left subtree.

Step 31: Recursively call bst_post_order on the right subtree.

Step 32: Print the value of the current node.

[End of function bst_post_order].

Delete Element (delete_elem)

Step 33: If the node is NULL, return NULL.

Step 34: Compare val with root->val. Perform the following:

a. If val is less, recursively call delete_elem on the left subtree.

b. If val is greater, recursively call delete_elem on the right subtree.

c. If val matches root->val, perform deletion:

i. If the node has no left child, return its right child.

ii. If the node has no right child, return its left child.

iii. If the node has two children, find the in-order successor, copy its value to root->val, and recursively delete the in-order successor.

Step 35: Return the modified root node.

[End of function delete_elem].

Main Menu (main)

Step 36: Display menu options to the user.

Step 37: Read the user's choice. If invalid, terminate with an error.

Step 38: Execute the appropriate function based on the choice.

Step 39: Repeat Step 36 until the user chooses to exit.

[End of loop defined at Step 36].

Step 40: Free memory allocated for the BST.

[End of function main]

Step 41: Stop.

Source Code

```
#include <stdio.h>
#include <stdlib.h>

typedef struct BST {
    int val;
    struct BST* lhs;
    struct BST* rhs;
} BST;

BST* _bst_insert_elem(BST* bst, int elem)
{
    if (bst == NULL) {
        bst = (BST*)malloc(sizeof(BST));
        if (bst == NULL) {
            fprintf(stderr, "error: malloc() failed.\n");
            exit(1);
        }

        bst->val = elem;
        bst->lhs = NULL;
        bst->rhs = NULL;

        return bst;
    }

    if (elem < bst->val) {
        bst->lhs = _bst_insert_elem(bst->lhs, elem);
    } else if (elem > bst->val) {
        bst->rhs = _bst_insert_elem(bst->rhs, elem);
    }

    return bst;
}

BST* bst_init(void)
{
    int val = 0;

    printf("Input data (-1 to quit): ");
    if (scanf("%d", &val) != 1) {
        fprintf(stderr, "error: Invalid input.\n");
        exit(1);
    }

    if (val == -1) {
        return NULL;
    }

    BST* bst = (BST*)malloc(sizeof(BST));
```

```
    if (bst == NULL) {
        fprintf(stderr, "error: malloc() failed.\n");
        exit(1);
    }

    bst->val = val;
    bst->lhs = NULL;
    bst->rhs = NULL;

    while (1) {
        printf("Input data (-1 to quit): ");
        if (scanf("%d", &val) != 1) {
            fprintf(stderr, "error: Invalid input.\n");
            exit(1);
        }

        if (val == -1) {
            break;
        }

        _bst_insert_elem(bst, val);
    }

    return bst;
}

void bst_insert_elem(BST* bst)
{
    int elem = 0;

    printf("Input an element to insert: ");
    if (scanf("%d", &elem) != 1) {
        fprintf(stderr, "error: Invalid input.\n");
        exit(1);
    }

    _bst_insert_elem(bst, elem);
}

void bst_pre_order(BST* bst)
{
    if (bst == NULL) {
        return;
    }

    printf("%d ", bst->val);
    bst_pre_order(bst->lhs);
    bst_pre_order(bst->rhs);
}
```

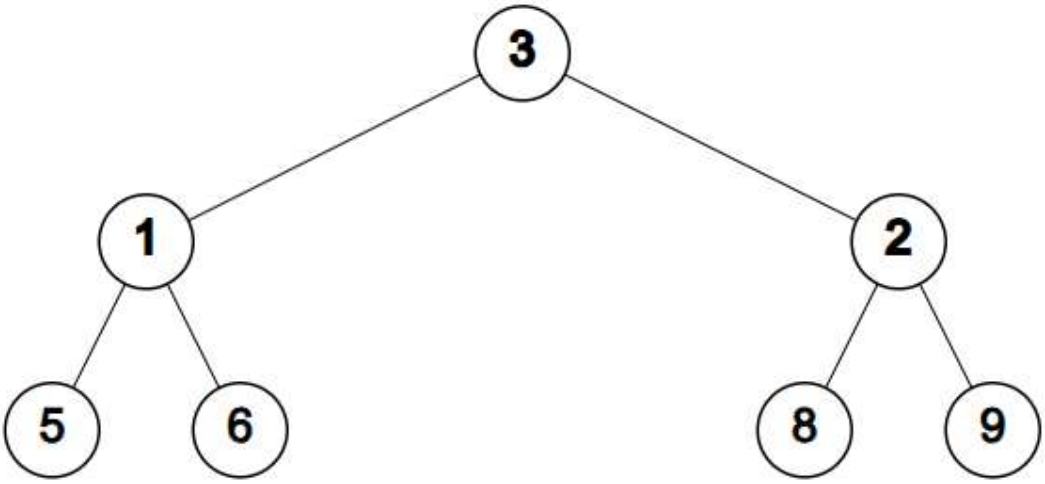
```
void bst_in_order(BST* bst)
{
    if (bst == NULL) {
        return;
    }

    bst_in_order(bst->lhs);
    printf("%d ", bst->val);
    bst_in_order(bst->rhs);
}

void bst_post_order(BST* bst)
{
    if (bst == NULL) {
        return;
    }

    bst_post_order(bst->lhs);
    bst_post_order(bst->rhs);
    printf("%d ", bst->val);
}

BST* find_min(BST* node)
{
    while (node->lhs != NULL) {
        node = node->lhs;
    }
    return node;
}

BST* delete_elem(BST* root, int val)
{
    if (root == NULL) {
        return root;
    }

    if (val < root->val) {
        root->lhs = delete_elem(root->lhs, val);
    } else if (val > root->val) {
        root->rhs = delete_elem(root->rhs, val);
    } else {
        // Node to be deleted found
        if (root->lhs == NULL) {
            BST* temp = root->rhs;
            free(root);
            return temp;
        } else if (root->rhs == NULL) {
            BST* temp = root->lhs;
            free(root);
            return temp;
```

```
        } else {
            // Node with two children
            BST* temp = find_min(root->rhs); // Find in-order successor
            root->val = temp->val; // Replace value
            root->rhs = delete_elem(root->rhs, temp->val); // Delete successor
        }
    }
    return root;
}

void bst_delete_elem(BST* bst)
{
    int val = 0;

    printf("Input an element to delete: ");
    if (scanf("%d", &val) != 1) {
        fprintf(stderr, "error: Invalid input.\n");
        exit(1);
    }

    bst = delete_elem(bst, val);
}

void bst_array_view(BST* bst)
{
    printf("Raw view: [ ");
    bst_in_order(bst);
    printf("]\n\n");
}

void bst_delete(BST* bst)
{
    if (bst == NULL) {
        return;
    }

    bst_delete(bst->lhs);
    bst_delete(bst->rhs);
    free(bst);
}

int main(void)
{
    int choice = 0;

    BST* bst = bst_init();

    do {
        printf("\n[1] Insert an element\n"
               "[2] Pre-order traversal\n"
```

```
                "[3] In-order traversal\n"
                "[4] Post-order traversal\n"
                "[5] Delete an element\n"
                "[6] Display nodes as it is\n"
                "[0] EXIT APPLICATION\n\n");

        printf("[ ] Your choice: ");
        if (scanf("%d", &choice) != 1) {
            fprintf(stderr, "error: Invalid input.\n");
            exit(1);
        }

        switch (choice) {
        case 0:
            goto quit;
            break;
        case 1:
            bst_insert_elem(bst);
            break;
        case 2:
            bst_pre_order(bst);
            printf("\n\n");
            break;
        case 3:
            bst_in_order(bst);
            printf("\n\n");
            break;
        case 4:
            bst_post_order(bst);
            printf("\n\n");
            break;
        case 5:
            bst_delete_elem(bst);
            break;
        case 6:
            bst_array_view(bst);
            break;
        default:
            printf("error: Invalid choice!\n");
            break;
        }
    } while (choice >= 0 && choice <= 7);

quit:
    printf("\n=== Thank you for using this app! ===\n");
    bst_delete(bst);

    return 0;
}
```

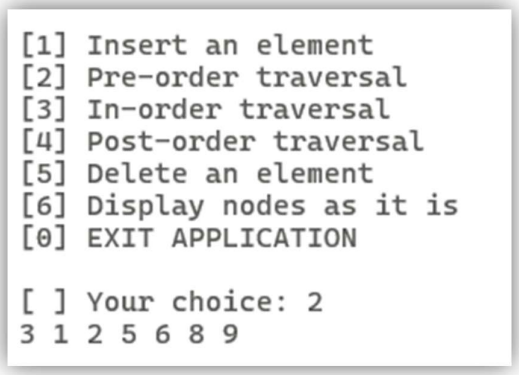
Output

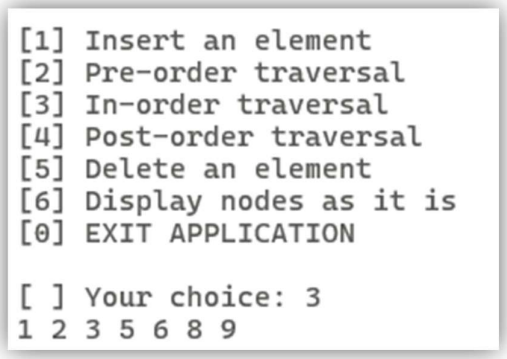
Input

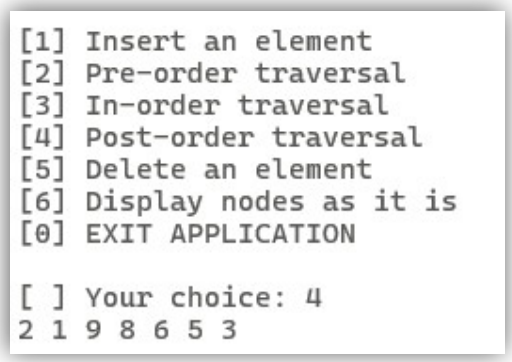


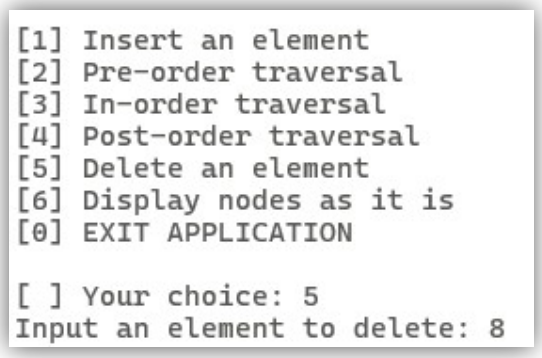

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.2314]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rohan\Code>gcc -g binary_search_tree.c

C:\Users\rohan\Code>a.exe
Input data (-1 to quit): 3
Input data (-1 to quit): 1
Input data (-1 to quit): 5
Input data (-1 to quit): 6
Input data (-1 to quit): 2
Input data (-1 to quit): 8
Input data (-1 to quit): 9
Input data (-1 to quit): -1
```

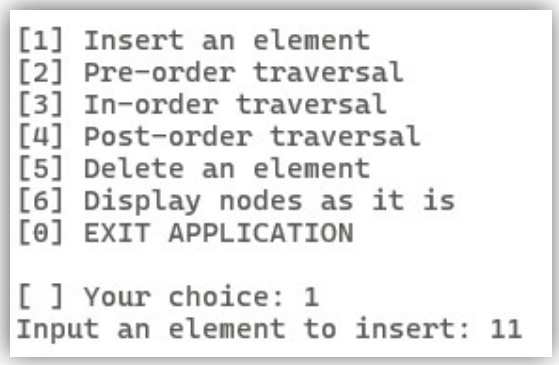
Operations

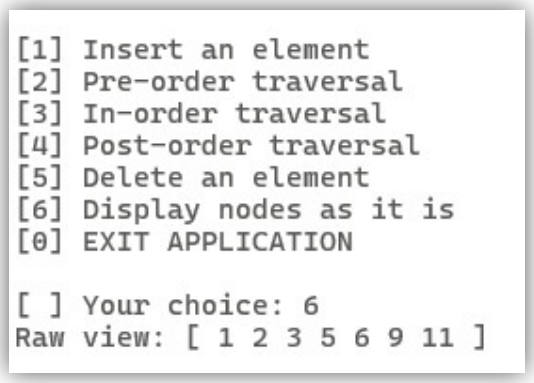












Discussion

Care should be taken while using a single pointer, as the pointer will itself pass by value, thus an easy trap. Use functions that returns pointer to make changes. Ensure user input validation and memory allocation.

Teacher's signature