Problem 8.c

```c
/**
 * Implementation for iterative traversal of a Simple Binary Tree
 * Author: Mithusayel Murmu
 */

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

/* Typedefs for BST and BSTNode */
typedef struct _BSTNode BSTNode;
typedef struct _BSTree BSTree;
/* Typedefs for Stack and StkNode */
typedef struct _StkNode StkNode;
typedef struct _Stack Stack;

/* Rudimentary Stack implementation */
struct _StkNode { BSTNode *data; StkNode *next; };
struct _Stack { int size; StkNode *head; };

Stack * stack_create() {
    Stack *stk = (Stack *) malloc(sizeof(Stack));
    stk->size = 0; stk->head = NULL;
    return stk;
}

void stack_destroy(Stack *stk) {
    if (!stk) return;

    StkNode *prev, *cur = stk->head;
    while (cur != NULL) { prev = cur; cur = cur->next; free(prev); }
    free(stk);
}

void stack_push(Stack *stk, BSTNode *data) {
    // Create a node
    StkNode *node = (StkNode *) malloc(sizeof(StkNode));
    node->data = data; node->next = NULL;

    if (stk->head == NULL) {
        stk->head = node;
    } else {
        node->next = stk->head;
        stk->head = node;
    }

    stk->size++;
}

BSTNode * stack_pop(Stack *stk) {
    if (!stk || stk->size == 0) return NULL;

    StkNode *temp = stk->head;
    BSTNode *ret = temp->data;
    stk->head = temp->next;
    stk->size--; free(temp);

    return ret;
}

BSTNode * stack_peek(const Stack *stk) {
    if (!stk || stk->size == 0) return NULL;
    return stk->head->data;
}

/* BST implementation */
/** Undiscovered, Discovered and Done */
typedef enum { WHITE, GRAY, BLACK } NodeState;
struct _BSTNode { int data; NodeState nstate; BSTNode *left, *right; };
struct _BSTree { int size; BSTNode *root; };

BSTree * bst_create() {
    BSTree *bst = (BSTree *) malloc(sizeof(BSTree));
    bst->size = 0; bst->root = NULL;
    return bst;
}

/**
 * Recursive definition for node insertion in BST
 * @node:   Pointer to BST's node pointer (typically the root)
```

```c
 * @data:   The data/value to insert in the tree
 */
void bst_insert_node(BSTNode **node, int data) {
    if (*node == NULL) {
        // Create new BSTNode
        BSTNode *_node = (BSTNode *) malloc(sizeof(BSTNode));
        _node->left = _node->right = NULL;
        _node->data = data; _node->nstate = WHITE;  // Undiscovered
        *node = _node;

        return;
    }

    // Redirect left
    if (data < (*node)->data)
        bst_insert_node(&(*node)->left, data);
    else  // Redirect right
        bst_insert_node(&(*node)->right, data);
}

void bst_reset_states(BSTNode *root) {
    if (!root) return;

    root->nstate = WHITE;  // Reset to undiscovered
    bst_reset_states(root->left);
    bst_reset_states(root->right);
}

/**
 * Iterative / Non-Recursive definition for inorder traversal of the BST
 * @tree:       Pointer to the BST to traverse
 * @callback:   Pointer to the callback function to process results
 */
void bst_traverse_in(const BSTree *tree, void (*callback)(int)) {
    if (!tree || !tree->size) return;

    BSTNode *pkN, *ppN;  // Peek node, Pop node
    Stack *stk = stack_create();
    // Push root in to the Stack
    stack_push(stk, tree->root);
    tree->root->nstate = GRAY;

    while (stk->size) {
        pkN = stack_peek(stk);
        // Push left if available and undiscovered
        if (pkN->left && pkN->left->nstate == WHITE) {
            stack_push(stk, pkN->left);
            pkN->left->nstate = GRAY;
        } else {
            ppN = stack_pop(stk);
            // Push right if available and undiscovered
            if (ppN->right && ppN->right->nstate == WHITE) {
                stack_push(stk, ppN->right);
                ppN->right->nstate = GRAY;
            }

            // Done with ppN. Evaluate current node
            ppN->nstate = BLACK;  callback(ppN->data);
        }
    }

    stack_destroy(stk); bst_reset_states(tree->root);
}

void bst_destroy_nodes(BSTNode **node) {
    if (*node == NULL) return;

    bst_destroy_nodes(&(*node)->left);
    bst_destroy_nodes(&(*node)->right);
    free(*node); *node = NULL;
}

void bst_insert(BSTree *bst, int data) {
    bst_insert_node(&bst->root, data);
    bst->size++;
}

void bst_destroy(BSTree *bst) {
    if (bst == NULL) return;
```

```c
    bst_destroy_nodes(&bst->root); free(bst);
}

void print_utility(int data) { printf("%d ", data); }
#define _scand(n) scanf("%d", &n)

int main(int argc, char const *argv[]) {
    int N, num;
    BSTree *bst = bst_create();

    printf("Number of elements to be inserted: ");
    _scand(N);
    printf("Enter %d space separated integers: ", N);

    while (N--) {
        _scand(num);
        bst_insert(bst, num);
    }

    printf("\nPrinting while traversal:\n");
    bst_traverse_in(bst, print_utility);
    printf("\n"); bst_destroy(bst);

    return 0;
}
```