**TO FIND LARGEST AND SMALLEST ELEMENT IN A BINARY SEARCH TREE**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*left, \*right;

};

struct node \*createnode(int key)

{

struct node \*newnode = (struct node\*)malloc(sizeof(struct node));

newnode->info = key;

newnode->left = NULL;

newnode->right = NULL;

return(newnode);

}

void inorder(struct node \*root)

{

if(root != NULL)

{

inorder(root->left);

printf(" %d ",root->info);

inorder(root->right);

}

}

void smallest(struct node \*root)

{

while(root != NULL && root->left != NULL)

{

root = root->left;

}

printf("\nSmallest value is %d\n", root->info);

}

void largest(struct node \*root)

{

while (root != NULL && root->right != NULL)

{

root = root->right;

}

printf("\nLargest value is %d", root->info);

}

/\*

\* Main Function

\*/

int main()

{

/\* Creating first Tree. \*/

struct node \*newnode = createnode(25);

newnode->left = createnode(17);

newnode->right = createnode(35);

newnode->left->left = createnode(13);

newnode->left->right = createnode(19);

newnode->right->left = createnode(27);

newnode->right->right = createnode(55);

/\* Sample Tree 1:

\* 25

\* / \

\* 17 35

\* / \ / \

\* 13 19 27 55

\*/

printf("Inorder traversal of tree 1 :");

inorder(newnode);

largest(newnode);

smallest(newnode);

**To Reverse the elements Singly Linked List**

#include <stdio.h>

#include <stdlib.h>

**/\* Link list node \*/**

struct Node {

int data;

struct Node\* next;

};

**/\* Function to reverse the linked list \*/**

static void reverse(struct Node\*\* head\_ref)

{

struct Node\* prev = NULL;

struct Node\* current = \*head\_ref;

struct Node\* next = NULL;

while (current != NULL) {

// Store next

next = current->next;

// Reverse current node's pointer

current->next = prev;

// Move pointers one position ahead.

prev = current;

current = next;

}

\*head\_ref = prev;

}

**/\* Function to push a node \*/**

void push(struct Node\*\* head\_ref, int new\_data)

{

/\* allocate node \*/

struct Node\* new\_node =

(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* link the old list of the new node \*/

new\_node->next = (\*head\_ref);

**/\* move the head to point to the new node \*/**

(\*head\_ref) = new\_node;

}

/\* Function to print linked list \*/

void printList(struct Node \*head)

{

struct Node \*temp = head;

while(temp != NULL)

{

printf("%d ", temp->data);

temp = temp->next;

}

}

int main()

{

/\* Start with the empty list \*/

struct Node\* head = NULL;

push(&head, 20);

push(&head, 4);

push(&head, 15);

push(&head, 85);

printf("Given linked list\n");

printList(head);

reverse(&head);

printf("\nReversed Linked list \n");

printList(head);

getchar();

}

**// C program for implementation of Bubble sort**

#include <stdio.h>

// Swap function

void swap(int\* arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

for (i = 0; i < n - 1; i++)

// Last i elements are already

// in place

for (j = 0; j < n - i - 1; j++)

if (arr[j] > arr[j + 1])

swap(arr, j, j + 1);

}

// Function to print an array

void printArray(int arr[], int size)

{

int i;

for (i = 0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver code

int main()

{

int arr[] = { 5, 1, 4, 2, 8 };

int N = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, N);

printf("Sorted array: ");

printArray(arr, N);

return 0;

}