DATA STRUCTURES LAB FINAL

**Name:** Ahmed Kasteer

**Roll Number**: 20F-0336

**Question 1:**

//QUESTION 1 LINKEDLIST

#include <iostream>

using namespace std;

struct Node

{

Node \* next;

int value;

};

Node\* head = NULL;

class Linkedlist

{

public:

Linkedlist()

{

int val = 0;

Node\* head = NULL;

}

~Linkedlist()

{

}

void insert(Node \* &head,int val)

{

Node \*newNode = new Node;

newNode->value = val;

if (head = NULL)

{

head = newNode;

}

newNode->next= head;

head = newNode;

if (val== newNode->value)

{

cout << "Value is already inserted into linkedlist and can't be duplicated" << endl;

}

}

void search(Node\* head, int val)

{

Node\* temp = head;

while (temp->next != NULL)

{

if (val == temp->value)

{

cout << "Value Found" << endl;

}

else

{

cout << "Value not found" << endl;

}

temp = temp->next;

}

}

void display(Node\* head)

{

Node\* temp = head;

while (temp->next != NULL)

{

cout << temp->value << endl;

temp = temp->next;

}

}

};

int main()

{

Linkedlist obj;

obj.insert(head,13);

obj.insert(head, 32);

obj.insert(head, 1);

obj.insert(head, 12);

obj.insert(head, 12);

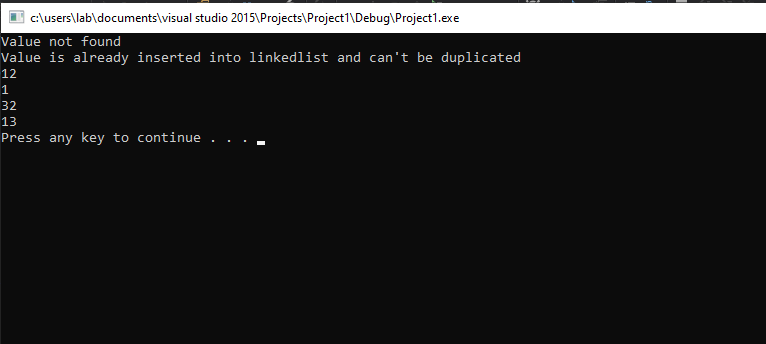
obj.search(head, 44);

obj.display(head);

system("pause");

return 0;

}

****

**Question 2:**

#include <iostream>

#include <string>

using namespace std;

class Stack

{

public:

string expression;

char value;

int top;

int size;

int \* stacksize = new int[size];

char arr[20];

Stack()

{

size = 0;

top = -1;

expression = "";

value = ' ';

}

bool isEmpty()

{

if (top == -1)

{

return true;

}

}

bool isFull()

{

if (top = size - 1)

{

return true;

}

}

void inputExpression()

{

cout << "Input expression" << endl;

cin >> arr[20];

for (int i = 0; i < 20; i++)

{

if (arr[i] == 'A' || arr[i] == 'B' || arr[i] == 'C' || arr[i] == 'D')

{

push(arr[i]);

}

else if (arr[i] == '+' || arr[i] == '-' || arr[i] == '\*' || arr[i] == '/')

{

pop(arr[i]);

pop(arr[i]);

}

expression = value;

push(expression);

}

}

void push(string val)

{

if (top != size -1)

{

stacksize[top] = val;

top++;

}

else

{

cout << "stack overflow" << endl;

}

}

void pop(string val)

{

if (top == -1)

{

cout << "Stack underflow" << endl;

}

val = stacksize[top];

top--;

}

void display()

{

for (int i = -1; i < size; i++)

{

cout << stacksize[i] << endl;

}

}

};

int main()

{

cout << "Input size" << endl;

Stack obj;

cin >> size;

cout << "Input expression" << endl;

getline(expression , cin);

obj.display();

system("pause");

return 0;

}

**Question 3:**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IMPORTANT INSTRUCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

// NOTE:- YOU SHOULD NOT ALTER THE PROVIDED CODE. JUST ADD YOUR LINES OF CODE /\*\*\*\* WRITE CODE HERE \*\*\*\*\*/ IS MENTIONED.

// IF YOU TRY TO CHANGE THE CODE, YOU WILL GET ZERO MARKS

// THE REQURIED LINES OF CODE IS ALSO PROVIDED WITH /\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

// DON'T EXCEED THE REQUIRED NUMBER OF LINES, WILL BE GET ZERO MARKS OTHERWISE.

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

/\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

// If left child is larger than root (~2 Lines Max.)

largest = (l < largest) ? l : largest;

/\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

// If right child is larger (~2 Lines Max.)

largest = (r > largest) ? r : largest;

/\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

// If largest is not root (~2 Lines Max.)

if (largest != i) {

largest = (l > r) ? l : r;

}// End if

} // End Heapify

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

// Move the Root to an end of array (In-Place Sorting)

//Rebuild the Heap

/\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

for (int i = n - 1; i > 0; i--) {

//(~ 2 Lines Max.)

i = i + 1;

heapify(arr, n, 1);

}

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

for (int i = 0; i < n; ++i)

cout << arr[i] << " ";

cout << "\n";

}

// MAIN METHOD

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

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

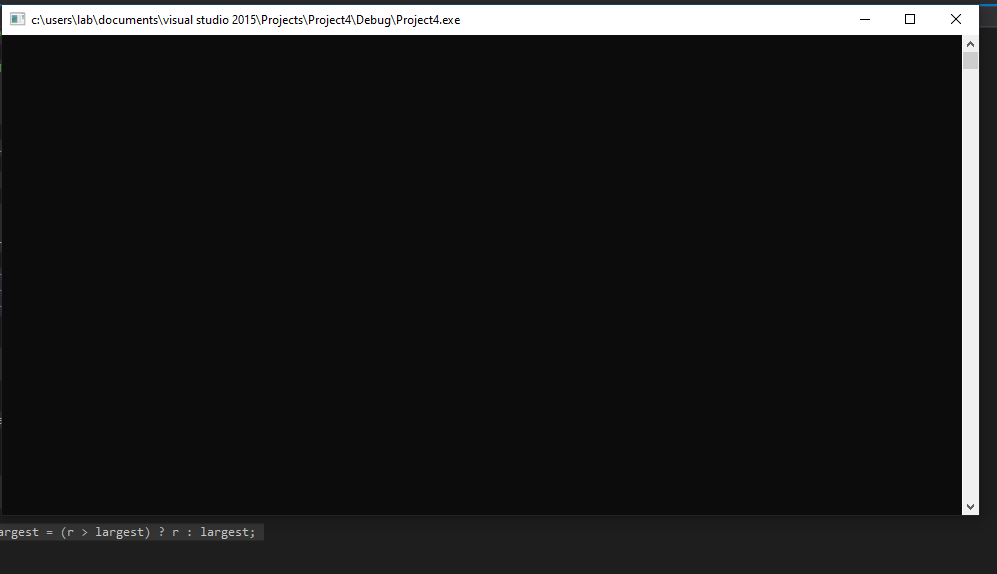
/\*\*\*\* WRITE CODE HERE \*\*\*\*\*/

// Call heap-sort routine nd rebuild the heap is required (~2 Lines Max.)

heapSort(arr, 5);

heapify(arr, 5,0);

}// END MAIN



Not displaying..

**Question 5:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* left;

int balancefactor;

int height;

node\* right;

};

class AVL

{

node \* root;

node\* getHeight(int a, int b)

{

if (a > b)

{

return;

}

}

node\* makeEmpty(node\* t) {

if (t == NULL)

return NULL;

{

makeEmpty(t->left);

makeEmpty(t->right);

delete t;

}

return NULL;

}

node\* insert(int x, node\* t)

{

if (t == NULL)

{

t = new node;

t->data = x;

t->left = t->right = NULL;

}

else if (x < t->data)

t->left = insert(x, t->left);

else if (x > t->data)

t->right = insert(x, t->right);

return t;

}

// now implementing rotations:

// right rotation with subtree managing

node \* rightrotation(node \* y)

{

node \* x = y->left;

node \* t2 = x->right;

x->right = y;

y->left = t2;

x->getHeight = int max(getHeight(x), getHeight(y)) + 1;

y->getHeight = int max(getHeight(y), getHeight(x)) + 1;

return y;

}

// single left rotation with subtree managing

node\* leftrotation(node\*x)

{

node\* y = x->right;

node\*t2 = y->left;

y->left = x;

x->right = t2;

y->getHeight = int max(getHeight(x), getHeight(y)) + 1;

x->getHeight = int max(getHeight(y), getHeight(x)) + 1;

}

//getting balanceFactor

node \* getBalanceFactor(node\*p)

{

if (p == NULL)

{

return p;

}

// right rotation

if (balancefactor > 1 && value < node->left->value)

{

rightrotation(node->right);

}

//left rotation

if (balancefactor < -1 && value > node->right value)

{

leftrotation(node->left);

}

// LR ROtation

if (balancefactor > 1 && value > node->left->value)

{

right->node = rightrotation(right rotation);

return leftrotation(left->rotation);

}

// RL Rotation

if (balancefactor < -1 && value < node->left->value)

{

left->node = rightrotation(left->rotation);

return rightrotation(right->rotation);

}

}

// checking balance factor

};

class BSTTOAVL {

node\* root;

node\* makeEmpty(node\* t) {

if (t == NULL)

return NULL;

{

makeEmpty(t->left);

makeEmpty(t->right);

delete t;

}

return NULL;

}

node\* insert(int x, node\* t)

{

if (t == NULL)

{

t = new node;

t->data = x;

t->left = t->right = NULL;

}

else if (x < t->data)

t->left = insert(x, t->left);

else if (x > t->data)

t->right = insert(x, t->right);

return t;

}

node\* findMin(node\* t)

{

if (t == NULL)

return NULL;

else if (t->left == NULL)

return t;

else

return findMin(t->left);

}

node\* findMax(node\* t) {

if (t == NULL)

return NULL;

else if (t->right == NULL)

return t;

else

return findMax(t->right);

}

node\* remove(int x, node\* t) {

node\* temp;

if (t == NULL)

return NULL;

else if (x < t->data)

t->left = remove(x, t->left);

else if (x > t->data)

t->right = remove(x, t->right);

else if (t->left && t->right)

{

temp = findMin(t->right);

t->data = temp->data;

t->right = remove(t->data, t->right);

}

else

{

temp = t;

if (t->left == NULL)

t = t->right;

else if (t->right == NULL)

t = t->left;

delete temp;

}

return t;

}

void inorder(node\* t) {

if (t == NULL)

return;

inorder(t->left);

cout << t->data << " ";

inorder(t->right);

}

node\* find(node\* t, int x) {

if (t == NULL)

return NULL;

else if (x < t->data)

return find(t->left, x);

else if (x > t->data)

return find(t->right, x);

else

return t;

}

public:

BSTTOAVL() {

root = NULL;

}

~BSTTOAVL() {

root = makeEmpty(root);

}

void insert(int x) {

root = insert(x, root);

}

void remove(int x) {

root = remove(x, root);

}

void display() {

inorder(root);

cout << endl;

}

void search(int x) {

root = find(root, x);

}

};

int main() {

BSTTOAVL t;

t.insert(20);

t.insert(25);

t.insert(15);

t.insert(10);

t.insert(30);

t.display();

t.remove(20);

t.display();

system("pause");

return 0;

}