Task 1

#include<iostream>

using namespace std;

class Node{

public:

int value;

Node\* left;

Node\* right;

Node(int value){

this->value =value;

this->left=NULL;

this->right=NULL;

}

};

class MaxHeap{

private:

Node\* root;

Node\* insert(Node\* root, int value){

if(root==NULL){

return new Node(value);

}

if(value>root->value){

root->left = insert(root->left,value);

}else{

root->right = insert(root->right,value);

}

return root;

}

Node\* Max(Node\* root){

while(root->right!=NULL){

root=root->right;

}

return root;

}

Node\* del(Node\* root,int value){

if(root==NULL){

return root;

}

if(value<root->value){

root->left=del(root->left,value);

}else if(value>root->value){

root->right=del(root->right,value);

}else{

if(root->left==NULL){

Node\* temp= root->right;

delete(root);

return temp;

}else if(root->right==NULL){

Node\* temp = root->left;

delete(root);

return temp;

}

////////

Node\* temp = Max(root->right);

root->value=temp->value;

root->right=del(root->right,temp->value);

}

return root;

}

void heapify(Node\* root){

if(root==NULL){

return;

}

Node\* largest = root;

Node\* left = root->left;

Node\* right = root->right;

if(left!=NULL&& left->value>largest->value){

largest=left;

}

if(right!=NULL&&right->value>largest->value){

largest=right;

}

if(largest!=root){

swap(root->value,largest->value);

heapify(largest);

}

}

void printHeap(Node\* root) {

if (root != NULL) {

printHeap(root->left);

cout << root->value << " ";

printHeap(root->right);

}

}

public:

MaxHeap():root(NULL){

}

void insert(int value){

root=insert(root,value);

}

void deleteKey(int value) {

root = del(root, value);

}

void heapify() {

heapify(root);

}

void printHeap() {

printHeap(root);

}

};

class MinHeap{

private:

Node\* root;

Node\* insert(Node\* root, int value){

if(root == NULL){

return new Node(value);

}

if(value<root->value){

root->left = insert(root->left,value);

}else{

root->right = insert(root->right,value);

}

//heapify(root);

return root;

}

Node\* findMin(Node\* root) {

while (root->left != NULL) {

root = root->left;

}

return root;

}

Node\* del(Node\* root , int value){

if(root == NULL){

return root;

}

if(value<root->value){

root->left = del(root->left,value);

}else if(value>root->value){

root->right = del(root->right,value);

}else{

if(root->left==NULL){

Node\* temp = root->right;

delete root ;

return temp;

}else if(root->right==NULL){

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMin(root->right);

root->value = temp->value;

root->right = del(root->right,temp->value);

}

return root;

}

void print(Node\* root) {

if (root != NULL) {

print(root->left);

cout << root->value << " ";

print(root->right);

}

}

void heapify(Node\* root) {

if (root == NULL) {

return;

}

Node\* smallest = root;

Node\* left = root->left;

Node\* right = root->right;

if (left != NULL && left->value < smallest->value) {

smallest = left;

}

if (right != NULL && right->value < smallest->value) {

smallest = right;

}

if (smallest != root) {

swap(root->value, smallest->value);

heapify(smallest);

}

}

public:

MinHeap(){

root = NULL;

}

void insert(int value) {

root = insert(root, value);

}

void deleteKey(int value) {

root = del(root, value);

}

void heapify() {

heapify(root);

}

void printHeap() {

print(root);

}

};

int main(){

MinHeap heap;

cout<<"Min heap"<<endl;

heap.insert(7);

heap.insert(1);

heap.insert(6);

heap.insert(2);

heap.insert(5);

heap.insert(9);

heap.insert(10);

heap.insert(2);

heap.printHeap();

cout<<endl<<"Max heap"<<endl;

MaxHeap mheap;

mheap.insert(7);

mheap.insert(1);

mheap.insert(6);

mheap.insert(2);

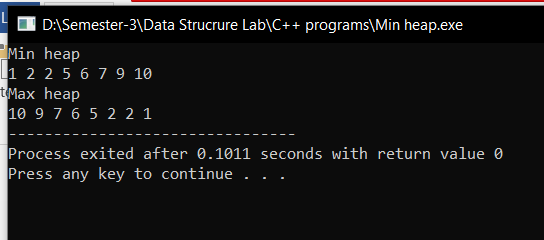
mheap.insert(5);

mheap.insert(9);

mheap.insert(10);

mheap.insert(2);

mheap.printHeap();

}

Task 2

#include<iostream>

using namespace std;

class Node{

public:

int value;

Node\* left;

Node\* right;

Node(int value){

this->value =value;

this->left=NULL;

this->right=NULL;

}

};

class MaxHeap{

private:

Node\* root;

Node\* insert(Node\* root, int value){

if(root==NULL){

return new Node(value);

}

if(value>root->value){

root->left = insert(root->left,value);

}else{

root->right = insert(root->right,value);

}

return root;

}

Node\* Max(Node\* root){

while(root->right!=NULL){

root=root->right;

}

return root;

}

Node\* del(Node\* root,int value){

if(root==NULL){

return root;

}

if(value<root->value){

root->left=del(root->left,value);

}else if(value>root->value){

root->right=del(root->right,value);

}else{

if(root->left==NULL){

Node\* temp= root->right;

delete(root);

return temp;

}else if(root->right==NULL){

Node\* temp = root->left;

delete(root);

return temp;

}

////////

Node\* temp = Max(root->right);

root->value=temp->value;

root->right=del(root->right,temp->value);

}

return root;

}

void heapify(Node\* root){

if(root==NULL){

return;

}

Node\* largest = root;

Node\* left = root->left;

Node\* right = root->right;

if(left!=NULL&& left->value>largest->value){

largest=left;

}

if(right!=NULL&&right->value>largest->value){

largest=right;

}

if(largest!=root){

swap(root->value,largest->value);

heapify(largest);

}

}

void printHeap(Node\* root) {

if (root != NULL) {

printHeap(root->left);

cout << root->value << " ";

printHeap(root->right);

}

}

public:

MaxHeap():root(NULL){

}

void insert(int value){

root=insert(root,value);

}

void deleteKey(int value) {

root = del(root, value);

}

void heapify() {

heapify(root);

}

void printHeap() {

printHeap(root);

}

};

class MinHeap{

private:

Node\* root;

Node\* insert(Node\* root, int value){

if(root == NULL){

return new Node(value);

}

if(value<root->value){

root->left = insert(root->left,value);

}else{

root->right = insert(root->right,value);

}

//heapify(root);

return root;

}

Node\* findMin(Node\* root) {

while (root->left != NULL) {

root = root->left;

}

return root;

}

Node\* del(Node\* root , int value){

if(root == NULL){

return root;

}

if(value<root->value){

root->left = del(root->left,value);

}else if(value>root->value){

root->right = del(root->right,value);

}else{

if(root->left==NULL){

Node\* temp = root->right;

delete root ;

return temp;

}else if(root->right==NULL){

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMin(root->right);

root->value = temp->value;

root->right = del(root->right,temp->value);

}

return root;

}

void print(Node\* root) {

if (root != NULL) {

print(root->left);

cout << root->value << " ";

print(root->right);

}

}

void heapify(Node\* root) {

if (root == NULL) {

return;

}

Node\* smallest = root;

Node\* left = root->left;

Node\* right = root->right;

if (left != NULL && left->value < smallest->value) {

smallest = left;

}

if (right != NULL && right->value < smallest->value) {

smallest = right;

}

if (smallest != root) {

swap(root->value, smallest->value);

heapify(smallest);

}

}

public:

MinHeap(){

root = NULL;

}

void insert(int value) {

root = insert(root, value);

}

void deleteKey(int value) {

root = del(root, value);

}

void heapify() {

heapify(root);

}

void printHeap() {

print(root);

}

};

int main(){

MinHeap heap;

int arr[]={35,33,42,10,14,19,27,44,26,31};

for(int i=0;i<10;i++){

heap.insert(arr[i]);

}

heap.printHeap();

cout<<"\nafter deleting 10";

heap.deleteKey(10);

cout<<endl;

heap.printHeap();

MaxHeap mheap;

int arrr[]={35,33,42,14,19,27,44,26,31};

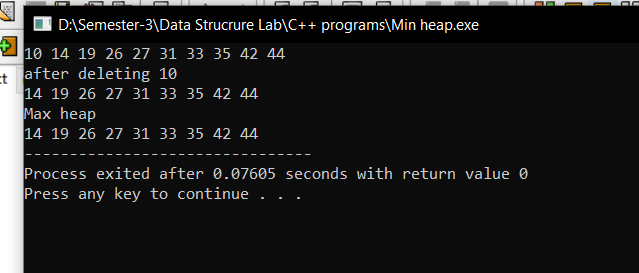
for(int i=0;i<10;i++){

mheap.insert(arrr[i]);

}

cout<<"\nMax heap"<<endl;

heap.printHeap();



Task 3

#include<iostream>

using namespace std;

class Node{

public:

int value;

Node\* left;

Node\* right;

Node(int v){

this->value = v;

this->left=NULL;

this->right=NULL;

}

};

class PriorityQueue{

private:

Node\* root;

void insert(Node\* &root,int value){

if(root==NULL){

root = new Node(value);

return;

}

if(root->left==NULL){

root->left=new Node(value);

}else if(root->right==NULL){

root->right = new Node(value);

}else{

insert(root->left,value);

}

}

void heapify(Node\* node) {

if (node == NULL) {

return;

}

Node\* smallest = node;

Node\* left = node->left;

Node\* right = node->right;

if (left != NULL && left->value < smallest->value) {

smallest = left;

}

if (right != NULL && right->value < smallest->value) {

smallest = right;

}

if (smallest != node) {

swap(node->value, smallest->value);

heapify(smallest);

}

}

public:

PriorityQueue(){

root=NULL;

}

void push(int value){

insert(root,value);

heapify(root);

}

void pop(){

if(root==NULL){

cout<<"Queue Emptyt"<<endl;

return;

}

Node\* rightLeaf = root;

while(rightLeaf->right!=NULL){

rightLeaf = rightLeaf->right;

}

swap(root->value,rightLeaf->value);

delete(rightLeaf);

}

int const top(){

if(root==NULL){

return -1;

}

return root->value;

}

bool const empty(){

return root==NULL;

}

};

int main(){

PriorityQueue q;

q.push(0);

q.push(5);

q.push(8);

q.push(6);

q.push(5);

q.push(4);

q.push(3);

q.push(1);

q.push(2);

while(q.empty()){

cout<<q.top()<<" ";

q.pop();

}

}