**DATA STRUCTURE**

**LAB 10**

**Name:**  Faizan Tariq

**Roll No:** 22F-3858

**TASK 1:**

#include<iostream>

using namespace std;

class Binary\_heap {

int\* arr;

int size;

int totalItems;

public:

Binary\_heap(int size) {

this->size = size;

totalItems = 0;

arr = new int[size];

}

int getMini(){

return arr[1];

}

void insert\_min\_heap(int number){

if(totalItems == size-1){

cout<<"Heap is full.\n";

return;

}

else{

++totalItems;

arr[totalItems] = number;

perculate\_up(totalItems);

}

}

void perculate\_up(int index){ // for min heap

while(index > 1 && arr[index/2] >= arr[index]){

swap(arr[index / 2], arr[index]);

index = index/2;

}

}

void perculate\_down(int index){

int smallerValue = index;

while(1){

if(index\*2 < size && arr[index\*2] < arr[smallerValue]){

smallerValue = index \* 2;

}

if(index\*2 +1 < size && arr[index\*2+1] < arr[smallerValue]){

smallerValue = index\*2 + 1;

}

if (smallerValue == index){

break;

}

swap(arr[smallerValue], arr[index]);

index= smallerValue;

}

}

void extractMin(){

cout << endl;

cout << arr[1] << " deleted.\n";

arr[1] = arr[totalItems--];

perculate\_down(1);

}

};

int main() {

Binary\_heap binary(100);

binary.insert\_min\_heap(4);

binary.insert\_min\_heap(1);

binary.insert\_min\_heap(8);

binary.insert\_min\_heap(12);

cout<<"GET MINI :"<<binary.getMini();

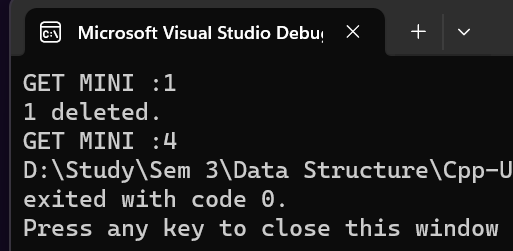
binary.extractMin();

cout << "GET MINI :" << binary.getMini();

return 0;

}

OUTPUT:



**TASK 2:**

#include<iostream>

using namespace std;

struct Building {

int id;

int floors;

Building() {

id = 0;

floors = 0;

}

};

class Binary\_heap {

Building\* arrayOfBuildings;

int size;

int totalBuilding;

public:

Binary\_heap(int size) {

this->size = size;

totalBuilding = 0;

arrayOfBuildings = new Building[size];

}

int getHighestPriorityBuilding() { // returns building with heighes priority

return arrayOfBuildings[1].id;

}

void insert\_min\_heap(int floor, int id) {

if (totalBuilding == size - 1) {

cout << "Heap is full.\n";

return;

}

else {

++totalBuilding;

arrayOfBuildings[totalBuilding].floors = floor;

arrayOfBuildings[totalBuilding].id= id;

perculate\_up(totalBuilding);

}

}

void perculate\_up(int index) { //

while (index > 1 && arrayOfBuildings[index / 2].floors <= arrayOfBuildings[index].floors) {

swap(arrayOfBuildings[index / 2].floors, arrayOfBuildings[index].floors);

swap(arrayOfBuildings[index / 2].id, arrayOfBuildings[index].id);

index = index / 2;

}

}

void perculate\_down(int index) {

int smallerValue = index;

while (1) {

if (index \* 2 < size && arrayOfBuildings[index \* 2].floors > arrayOfBuildings[smallerValue].floors) {

smallerValue = index \* 2;

}

if (index \* 2 + 1 < size && arrayOfBuildings[index \* 2 + 1].floors > arrayOfBuildings[smallerValue].floors) {

smallerValue = index \* 2 + 1;

}

if (smallerValue == index) {

break;

}

swap(arrayOfBuildings[smallerValue].floors, arrayOfBuildings[index].floors);

swap(arrayOfBuildings[smallerValue].id, arrayOfBuildings[index].id);

index = smallerValue;

}

}

void constructBuilding() {

cout << endl;

cout << arrayOfBuildings[1].id << " deleted having \"" << arrayOfBuildings[1].floors<<"\" floors";

arrayOfBuildings[1] = arrayOfBuildings[totalBuilding--];

perculate\_down(1);

}

};

int main() {

Binary\_heap binary(100);

binary.insert\_min\_heap(20,1);

binary.insert\_min\_heap(45, 2);

cout << "\nGET MINI :" << binary.getHighestPriorityBuilding();

binary.insert\_min\_heap(5,3);

cout << "\nGET MINI :" << binary.getHighestPriorityBuilding();

binary.constructBuilding();

cout << "\nGET MINI :" << binary.getHighestPriorityBuilding();

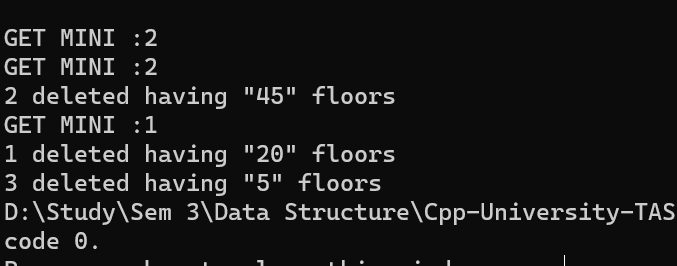
binary.constructBuilding();

binary.constructBuilding();

return 0;

}

**OUTPUT:**

****

**TASK 3:**

##include<iostream>

using namespace std;

struct Node {

int data;

Node\* left, \* right;

Node() {

data = 0;

left = NULL;

right = NULL;

}

};

class BST {

int start;

public:

Node\* root;

BST() {

root = NULL;

start = -1;

}

Node\* insertNode(Node\* Root, int num) {

Node\* newNode = new Node;

newNode->data = num;

if (!Root) {

Root = newNode;

}

else {

if (num < Root->data) {

Root->left = insertNode(Root->left, num);

}

else if (num > Root->data) {

Root->right = insertNode(Root->right, num);

}

}

return Root;

}

void inOrder(Node\* Root, int arr[]) {

if (Root) { // Changed 'if (root)' to 'if (Root)'

inOrder(Root->left, arr);

arr[++start] = Root->data;

inOrder(Root->right, arr);

}

}

Node\* getRoot() {

return root;

}

};

int main() {

BST tree;

tree.root = tree.insertNode(tree.getRoot(), 1); // Assign root after inserting the first node

tree.insertNode(tree.getRoot(), 3);

tree.insertNode(tree.getRoot(), 2);

tree.insertNode(tree.getRoot(), 9);

tree.insertNode(tree.getRoot(), 7);

int arr[5] = {};

tree.inOrder(tree.getRoot(), arr);

cout << endl;

for (int i = 0; i < 5; i++) {

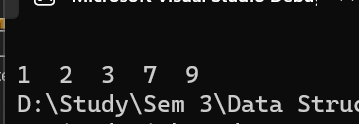
cout << arr[i] << " ";

}

return 0;

}

**OUTPUT**

****

**TASK 4:**#include<iostream>

using namespace std;

class Binary\_heap {

int\* arr;

int size;

int totalItems;

public:

Binary\_heap(int size) {

this->size = size;

totalItems = 0;

arr = new int[size];

}

int getMini() {

return arr[1];

}

void insert\_max\_heap(int number) {

if (totalItems == size - 1) {

cout << "Heap is full.\n";

return;

}

else {

++totalItems;

arr[totalItems] = number;

percolate\_up\_max(totalItems);

}

}

void percolate\_up\_max(int index) {

while (index > 1 && arr[index / 2] < arr[index]) {

swap(arr[index / 2], arr[index]);

index = index / 2;

}

}

void percolate\_down\_max(int index) {

int largerValue = index;

while (1) {

if (index \* 2 <= totalItems && arr[index \* 2] >= arr[largerValue]) {

largerValue = index \* 2;

}

if (index \* 2 + 1 <= totalItems && arr[index \* 2 + 1] >= arr[largerValue]) {

largerValue = index \* 2 + 1;

}

if (largerValue == index) {

break;

}

swap(arr[largerValue], arr[index]);

index = largerValue;

}

}

void extractMax(int& deletedNum) {

cout << arr[1] << " deleted.\n";

deletedNum = arr[1];

arr[1] = arr[totalItems];

totalItems--;

percolate\_down\_max(1);

}

};

int main() {

Binary\_heap heapmax(100);

int arr[10] = { 3, 5, 9, 6, 8, 20, 10, 12, 18, 9 };

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

heapmax.insert\_max\_heap(arr[i]);

}

int deletedNum = 0;

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

heapmax.extractMax(deletedNum);

arr[i] = deletedNum;

}

cout << "RESULT ARRAY : \n";

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

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

}

return 0;

}

**OUTPUT:**

**A black screen with white text

Description automatically generated**