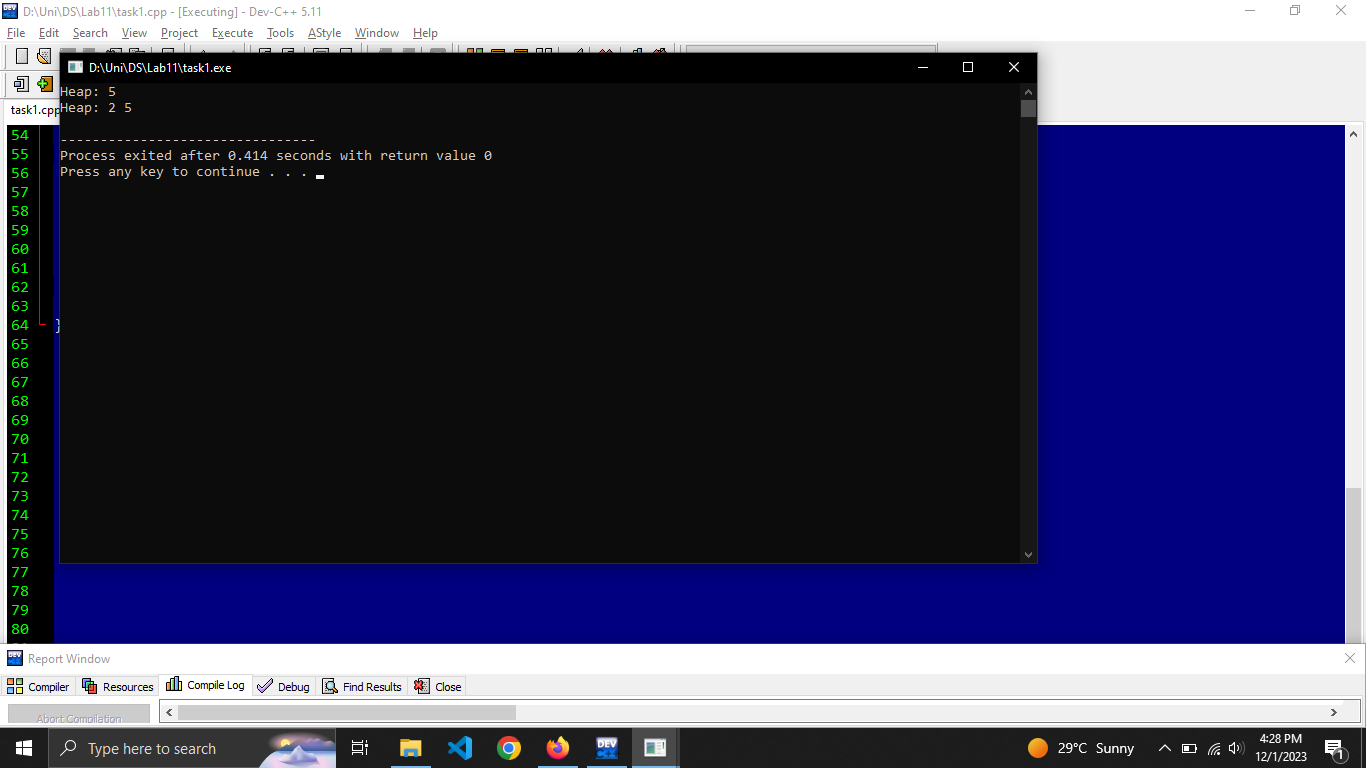
**DS LAB 11**

**K226007**

**Task 1:**



**#include <iostream>**

**#define MAX\_SIZE 100**

**using namespace std;**

**class MinHeap**

**{**

**private:**

**int heap[MAX\_SIZE];**

**int size;**

**void heapifyUp(int index)**

**{**

**while (index > 0)**

**{**

**int parentIndex = (index - 1) / 2;**

**if (heap[index] < heap[parentIndex])**

**{**

**swap(heap[index], heap[parentIndex]);**

**index = parentIndex;**

**} else**

**{**

**break;**

**}**

**}**

**}**

**public:**

**MinHeap() : size(0) {}**

**void insert(int priority)**

**{**

**if (size < MAX\_SIZE)**

**{**

**heap[size++] = priority;**

**heapifyUp(size - 1);**

**} else**

**{**

**cout << "Heap is full. Cannot insert more elements." << endl;**

**}**

**}**

**void displayHeap()**

**{**

**cout << "Heap: ";**

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

**{**

**cout << heap[i] << " ";**

**}**

**cout << endl;**

**}**

**};**

**int main() {**

**MinHeap heap;**

**heap.insert(5);**

**heap.displayHeap();**

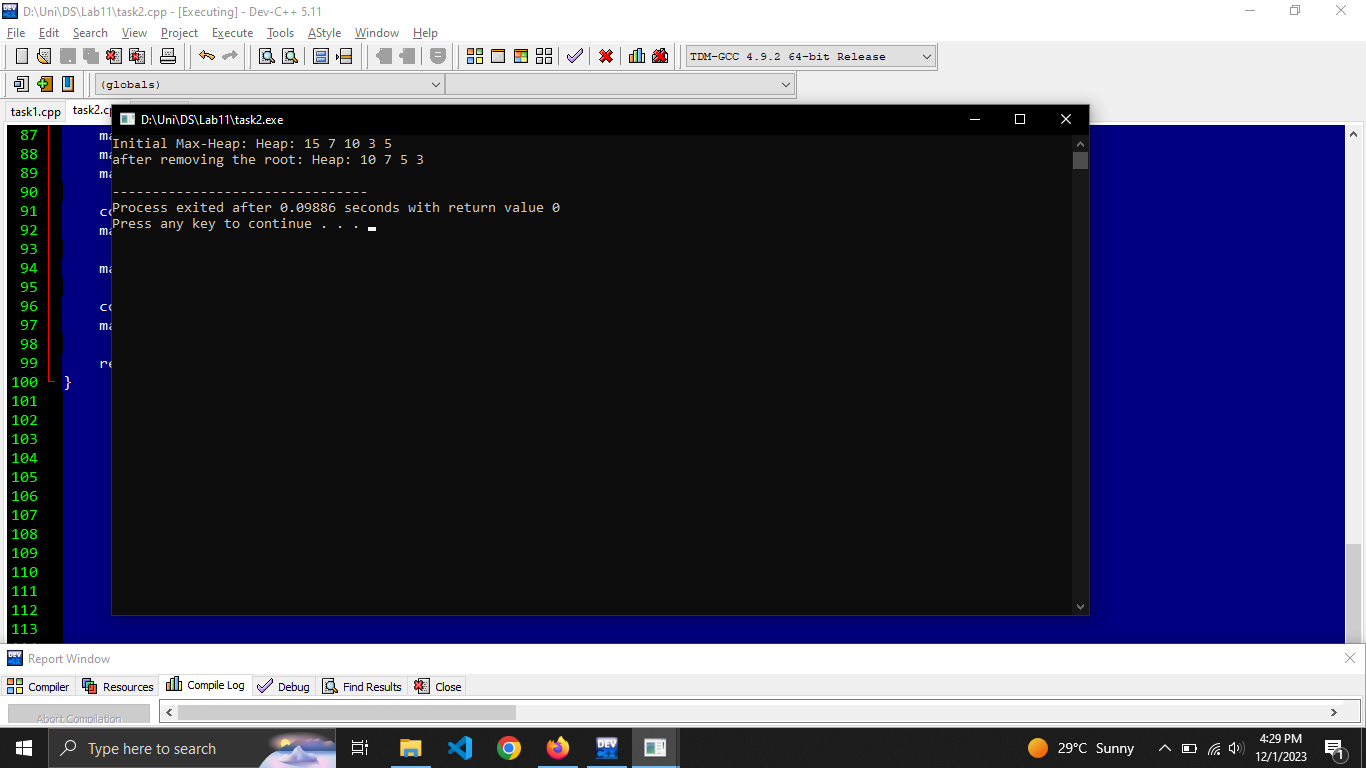
**heap.insert(2);**

**heap.displayHeap();**

**return 0;**

**}**

**Task2:**



**#include <iostream>**

**#define MAX\_SIZE 100**

**using namespace std;**

**class MaxHeap**

**{**

**private:**

**int heap[MAX\_SIZE];**

**int size;**

**void heapifyDown(int index)**

**{**

**int leftChild = 2 \* index + 1;**

**int rightChild = 2 \* index + 2;**

**int largest = index;**

**if (leftChild < size && heap[leftChild] > heap[largest])**

**{**

**largest = leftChild;**

**}**

**if (rightChild < size && heap[rightChild] > heap[largest])**

**{**

**largest = rightChild;**

**}**

**if (largest != index)**

**{**

**swap(heap[index], heap[largest]);**

**heapifyDown(largest);**

**}**

**}**

**public:**

**MaxHeap() : size(0) {}**

**void insert(int priority)**

**{**

**if (size < MAX\_SIZE)**

**{**

**heap[size++] = priority;**

**int currentIndex = size - 1;**

**int parentIndex = (currentIndex - 1) / 2;**

**while (currentIndex > 0 && heap[currentIndex] > heap[parentIndex])**

**{**

**swap(heap[currentIndex], heap[parentIndex]);**

**currentIndex = parentIndex;**

**parentIndex = (currentIndex - 1) / 2;**

**}**

**} else**

**{**

**cout << "Heap is full." << endl;**

**}**

**}**

**void removeRoot()**

**{**

**if (size > 0)**

**{**

**heap[0] = heap[size - 1];**

**size--;**

**heapifyDown(0);**

**} else {**

**cout << "Heap is empt." << endl;**

**}**

**}**

**void displayHeap()**

**{**

**cout << "Heap: ";**

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

**{**

**cout << heap[i] << " ";**

**}**

**cout << endl;**

**}**

**};**

**int main() {**

**MaxHeap maxHeap;**

**maxHeap.insert(10);**

**maxHeap.insert(5);**

**maxHeap.insert(15);**

**maxHeap.insert(3);**

**maxHeap.insert(7);**

**cout << "Initial Max-Heap: ";**

**maxHeap.displayHeap();**

**maxHeap.removeRoot();**

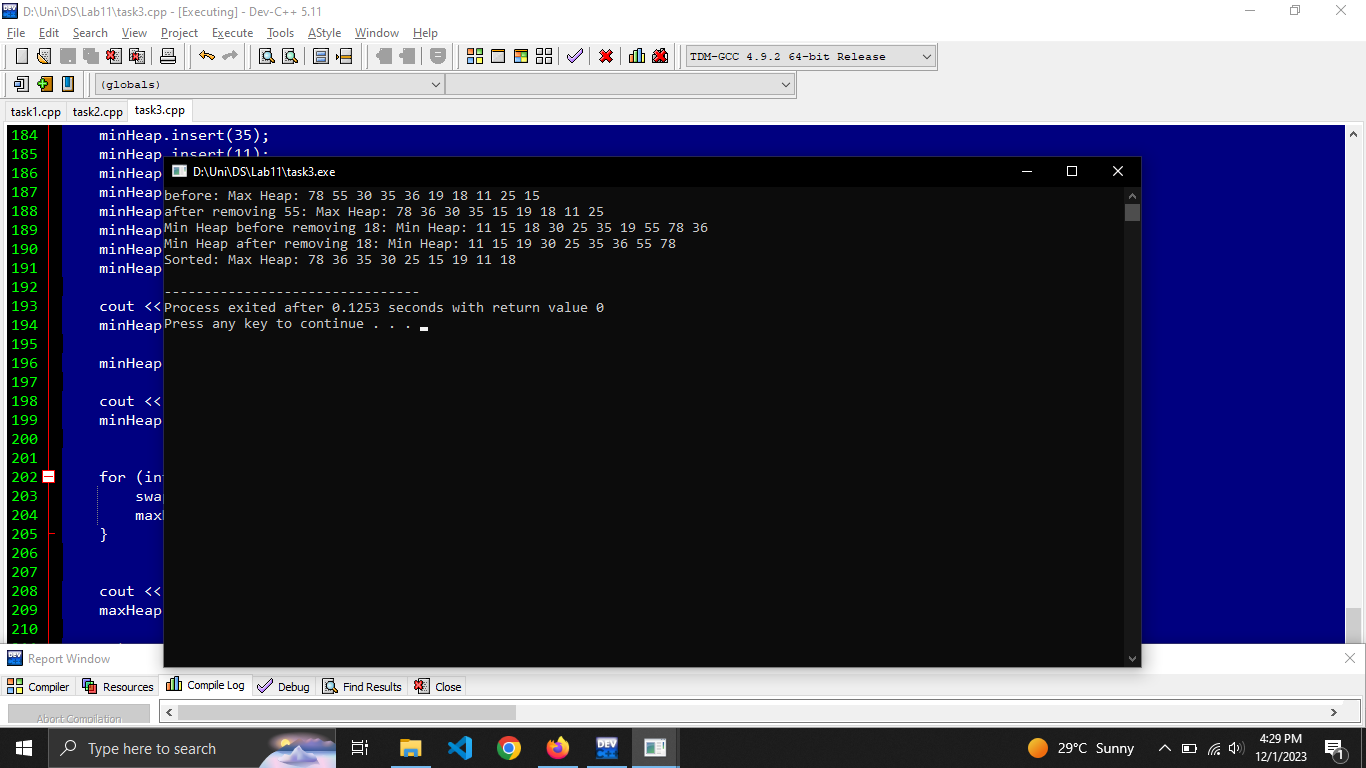
**cout << "after removing the root: ";**

**maxHeap.displayHeap();**

**return 0;**

**}**

**Task 3:**



**#include <iostream>**

**using namespace std;**

**class MaxHeap {**

**public:**

**static const int MAX\_SIZE = 100;**

**int heap[MAX\_SIZE];**

**int size;**

**void heapifyDown(int index) {**

**int leftChild = 2 \* index + 1;**

**int rightChild = 2 \* index + 2;**

**int largest = index;**

**if (leftChild < size && heap[leftChild] > heap[largest])**

**{**

**largest = leftChild;**

**}**

**if (rightChild < size && heap[rightChild] > heap[largest])**

**{**

**largest = rightChild;**

**}**

**if (largest != index)**

**{**

**swap(heap[index], heap[largest]);**

**heapifyDown(largest);**

**}**

**}**

**public:**

**MaxHeap() : size(0) {}**

**void insert(int value)**

**{**

**if (size < MAX\_SIZE)**

**{**

**heap[size++] = value;**

**int currentIndex = size - 1;**

**int parentIndex = (currentIndex - 1) / 2;**

**while (currentIndex > 0 && heap[currentIndex] > heap[parentIndex])**

**{**

**swap(heap[currentIndex], heap[parentIndex]);**

**currentIndex = parentIndex;**

**parentIndex = (currentIndex - 1) / 2;**

**}**

**} else {**

**cout << "Heap is full." << endl;**

**}**

**}**

**void removeValue(int value)**

**{**

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

**{**

**if (heap[i] == value)**

**{**

**heap[i] = heap[size - 1];**

**size--;**

**heapifyDown(i);**

**break;**

**}**

**}**

**}**

**void displayHeap() {**

**cout << "Max Heap: ";**

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

**{**

**cout << heap[i] << " ";**

**}**

**cout << endl;**

**}**

**};**

**class MinHeap**

**{**

**private:**

**static const int MAX\_SIZE = 100;**

**int heap[MAX\_SIZE];**

**int size;**

**void heapifyDown(int index)**

**{**

**int leftChild = 2 \* index + 1;**

**int rightChild = 2 \* index + 2;**

**int smallest = index;**

**if (leftChild < size && heap[leftChild] < heap[smallest])**

**{**

**smallest = leftChild;**

**}**

**if (rightChild < size && heap[rightChild] < heap[smallest])**

**{**

**smallest = rightChild;**

**}**

**if (smallest != index)**

**{**

**swap(heap[index], heap[smallest]);**

**heapifyDown(smallest);**

**}**

**}**

**public:**

**MinHeap() : size(0) {}**

**void insert(int value)**

**{**

**if (size < MAX\_SIZE)**

**{**

**heap[size++] = value;**

**int currentIndex = size - 1;**

**int parentIndex = (currentIndex - 1) / 2;**

**while (currentIndex > 0 && heap[currentIndex] < heap[parentIndex])**

**{**

**swap(heap[currentIndex], heap[parentIndex]);**

**currentIndex = parentIndex;**

**parentIndex = (currentIndex - 1) / 2;**

**}**

**} else**

**{**

**cout << "Heap is full." << endl;**

**}**

**}**

**void removeValue(int value)**

**{**

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

**{**

**if (heap[i] == value)**

**{**

**heap[i] = heap[size - 1];**

**size--;**

**heapifyDown(i);**

**break;**

**}**

**}**

**}**

**void displayHeap() {**

**cout << "Min Heap: ";**

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

**{**

**cout << heap[i] << " ";**

**}**

**cout << endl;**

**}**

**};**

**int main()**

**{**

**MaxHeap maxHeap;**

**maxHeap.insert(25);**

**maxHeap.insert(30);**

**maxHeap.insert(35);**

**maxHeap.insert(11);**

**maxHeap.insert(15);**

**maxHeap.insert(19);**

**maxHeap.insert(18);**

**maxHeap.insert(55);**

**maxHeap.insert(78);**

**maxHeap.insert(36);**

**cout << "before: ";**

**maxHeap.displayHeap();**

**maxHeap.removeValue(55);**

**cout << "after removing 55: ";**

**maxHeap.displayHeap();**

**MinHeap minHeap;**

**minHeap.insert(25);**

**minHeap.insert(30);**

**minHeap.insert(35);**

**minHeap.insert(11);**

**minHeap.insert(15);**

**minHeap.insert(19);**

**minHeap.insert(18);**

**minHeap.insert(55);**

**minHeap.insert(78);**

**minHeap.insert(36);**

**cout << "Min Heap before removing 18: ";**

**minHeap.displayHeap();**

**minHeap.removeValue(18);**

**cout << "Min Heap after removing 18: ";**

**minHeap.displayHeap();**

**for (int i = maxHeap.size - 1; i > 0; --i) {**

**swap(maxHeap.heap[0], maxHeap.heap[i]);**

**maxHeap.heapifyDown(0);**

**}**

**cout << "Sorted: ";**

**maxHeap.displayHeap();**

**return 0;**

**}**