Code for Binary Heap

#include<iostream>

#include<climits>

using namespace std;

// Prototype of a utility function to swap two integers

void swap(int \*x, int \*y);

// A class for Min Heap

class MinHeap

{

    int \*harr; // pointer to array of elements in heap

    int capacity; // maximum possible size of min heap

    int heap\_size; // Current number of elements in min heap

public:

    // Constructor

    MinHeap(int capacity);

    // to heapify a subtree with the root at given index

    void MinHeapify(int );

    int parent(int i) { return (i-1)/2; }

    // to get index of left child of node at index i

    int left(int i) { return (2\*i + 1); }

    // to get index of right child of node at index i

    int right(int i) { return (2\*i + 2); }

    // to extract the root which is the minimum element

    int extractMin();

    // Decreases key value of key at index i to new\_val

    void decreaseKey(int i, int new\_val);

    // Returns the minimum key (key at root) from min heap

    int getMin() { return harr[0]; }

    // Deletes a key stored at index i

    void deleteKey(int i);

    // Inserts a new key 'k'

    void insertKey(int k);

};

// Constructor: Builds a heap from a given array a[] of given size

MinHeap::MinHeap(int cap)

{

    heap\_size = 0;

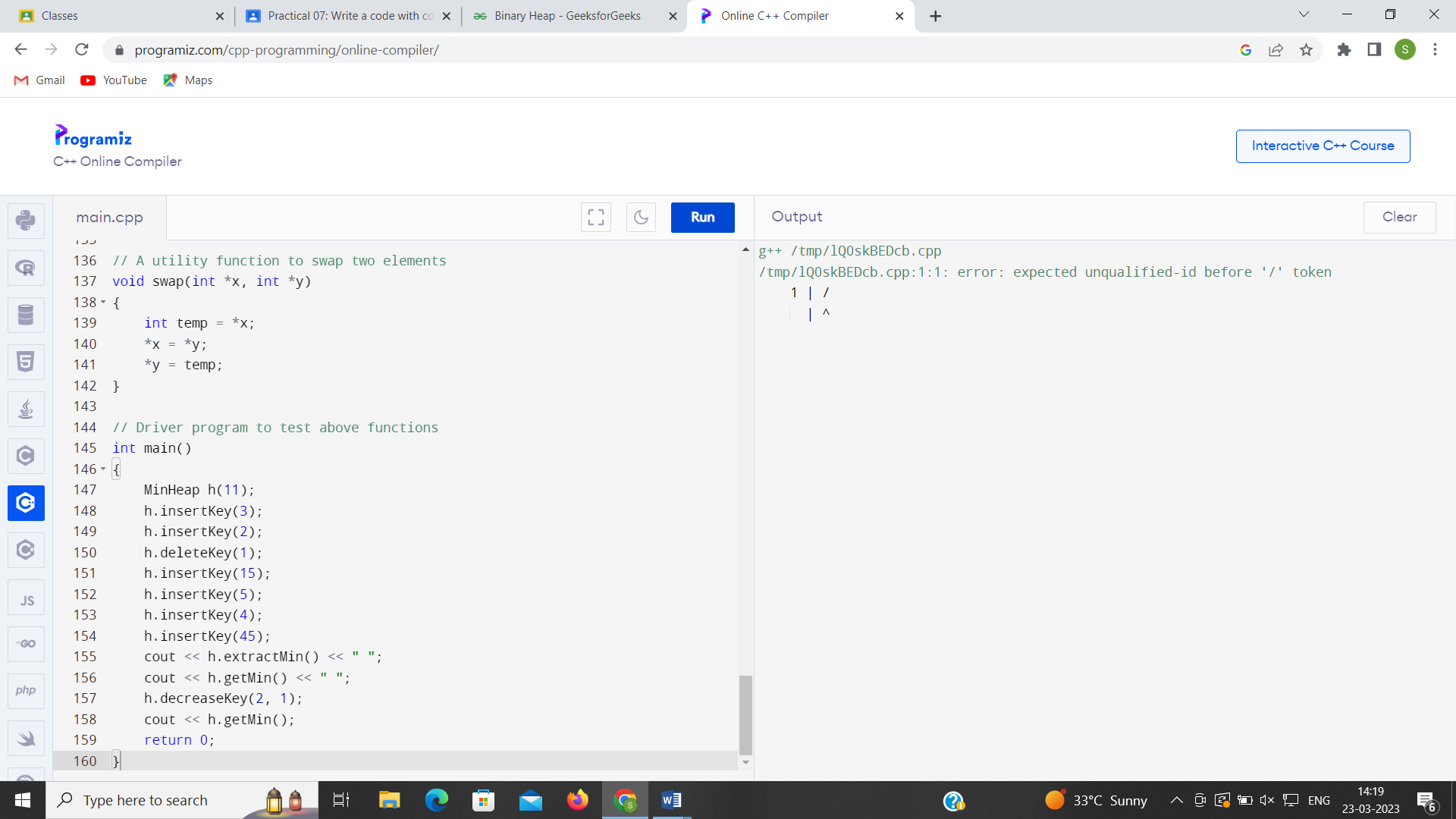
    capacity = cap;

    harr = new int[cap];

}

// Inserts a new key 'k'

void MinHeap::insertKey(int k)



{

    if (heap\_size == capacity)

    {

        cout << "\nOverflow: Could not insertKey\n";

        return;

    }

    // First insert the new key at the end

    heap\_size++;

    int i = heap\_size - 1;

    harr[i] = k;

    // Fix the min heap property if it is violated

    while (i != 0 && harr[parent(i)] > harr[i])

    {

       swap(&harr[i], &harr[parent(i)]);

       i = parent(i);

    }

}

// Decreases value of key at index 'i' to new\_val.  It is assumed that

// new\_val is smaller than harr[i].

void MinHeap::decreaseKey(int i, int new\_val)

{

    harr[i] = new\_val;

    while (i != 0 && harr[parent(i)] > harr[i])

    {

       swap(&harr[i], &harr[parent(i)]);

       i = parent(i);

    }

}

// Method to remove minimum element (or root) from min heap

int MinHeap::extractMin()

{

    if (heap\_size <= 0)

        return INT\_MAX;

    if (heap\_size == 1)

    {

        heap\_size--;

        return harr[0];

    }

    // Store the minimum value, and remove it from heap

    int root = harr[0];

    harr[0] = harr[heap\_size-1];

    heap\_size--;

    MinHeapify(0);

    return root;

}

// This function deletes key at index i. It first reduced value to minus

// infinite, then calls extractMin()

void MinHeap::deleteKey(int i)

{

    decreaseKey(i, INT\_MIN);

    extractMin();

}

// A recursive method to heapify a subtree with the root at given index

// This method assumes that the subtrees are already heapified

void MinHeap::MinHeapify(int i)

{

    int l = left(i);

    int r = right(i);

    int smallest = i;

    if (l < heap\_size && harr[l] < harr[i])

        smallest = l;

    if (r < heap\_size && harr[r] < harr[smallest])

        smallest = r;

    if (smallest != i)

    {

        swap(&harr[i], &harr[smallest]);

        MinHeapify(smallest);

    }

}

// A utility function to swap two elements

void swap(int \*x, int \*y)

{

    int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

// Driver program to test above functions

int main()

{

    MinHeap h(11);

    h.insertKey(3);

    h.insertKey(2);

    h.deleteKey(1);

    h.insertKey(15);

    h.insertKey(5);

    h.insertKey(4);

    h.insertKey(45);

    cout << h.extractMin() << " ";

    cout << h.getMin() << " ";

    h.decreaseKey(2, 1);

    cout << h.getMin();

    return 0;

}