TASK 01

#include <iostream>

using namespace std;

void swap(int& a,int& b){

    int tmp=a;

    a=b;

    b=tmp;

}

void display(int arr[], int size){

    for (int i=0; i<size; ++i) cout<<arr[i]<<" ";

}

void heapify\_down(int arr[],int size,int index,bool max) {

    int leftChild=2\*index+1;

    int rightChild=2\*index+2;

    int tmpIndex=index;

    if(max){

        if (leftChild<size && arr[leftChild]>arr[tmpIndex]) tmpIndex = leftChild;

        if (rightChild<size && arr[rightChild]>arr[tmpIndex]) tmpIndex = rightChild;

    } else {

        if (leftChild<size && arr[leftChild]<arr[tmpIndex]) tmpIndex = leftChild;

        if (rightChild<size && arr[rightChild]<arr[tmpIndex]) tmpIndex = rightChild;

    }

    if (tmpIndex != index) {

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

        heapify\_down(arr,size,tmpIndex,max);

    }

}

void buildHeap(int arr[],int size,bool max){

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

        heapify\_down(arr,size,i,max);

    }

}

int main(){

    int arr[]={7,1,6,3,5,9,10,2,4,8};

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

    cout<<"Orignal  : ";

    display(arr,size);

    cout<<endl;

    cout<<"Max Heap : ";

    buildHeap(arr,size,true);

    display(arr,size);

    cout<<endl;

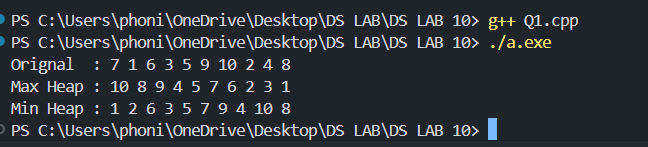
    cout<<"Min Heap : ";

    buildHeap(arr,size,false);

    display(arr,size);

    return 0;

}



TASK 02

#include <iostream>

using namespace std;

void swap(int& a,int& b){

    int tmp=a;

    a=b;

    b=tmp;

}

class Heap{

    private:

        int arr[100];

        int pos;

        bool type;

    public:

        Heap(bool \_type):type(\_type),pos(-1){};

        Heap(const Heap& obj) : type(obj.type), pos(obj.pos) {

            for (int i=0; i<=pos; i++) this->arr[i]=obj.arr[i];

        }

        void heapify\_up(int index){

            if(index<1) return;

            int parent=(index-1)/2;

            if(type && arr[parent]<arr[index]){

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

                heapify\_up(parent);

            } else if(!type && arr[parent]>arr[index]){

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

                heapify\_up(parent);

            }

        }

        void heapify\_down(int index){

            int leftChild=2\*index+1;

            int rightChild=2\*index+2;

            int minIndex=index;

            if(type){

                if(leftChild<=pos && arr[leftChild]>arr[minIndex]) minIndex=leftChild;

                if(rightChild<=pos && arr[rightChild]>arr[minIndex]) minIndex=rightChild;

            } else{

                if(leftChild<=pos && arr[leftChild]<arr[minIndex]) minIndex=leftChild;

                if(rightChild<=pos && arr[rightChild]<arr[minIndex]) minIndex=rightChild;

            }

            if(minIndex!=index){

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

                heapify\_down(minIndex);

            }

        }

        void changeType(bool type){

            this->type=type;

            for(int i=(pos+2)/2 ; i>=0 ; i--){

                heapify\_down(i);

            }

        }

        void insert(int val){

            if(pos+1>=100) return;

            arr[++pos]=val;

            heapify\_up(pos);

        }

        int remove(int index){

            if(pos<0 || index<0 || index>pos) return -1;

            int tmp=arr[index];

            arr[index]=arr[pos--];

            heapify\_down(index);

            return tmp;

        }

        void printHeap() {

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

                cout<<arr[i]<<" ";

            }

            cout<<endl;

        }

        bool isEmpty(){

            return pos<0;

        }

};

int main(){

    Heap heap(true);

    heap.insert(35);

    heap.insert(33);

    heap.insert(42);

    heap.insert(10);

    heap.insert(14);

    heap.insert(19);

    heap.insert(27);

    heap.insert(44);

    heap.insert(26);

    heap.insert(31);

    cout<<"Min Heap"<<endl;

    heap.printHeap();

    cout<<endl;

    heap.remove(0);

    cout<<"After deleting the root. Max Heap"<<endl;

    heap.changeType(false);

    heap.printHeap();

    cout<<endl;

    cout<<"Sorted Manner: "<<endl;

    Heap tmp=heap;

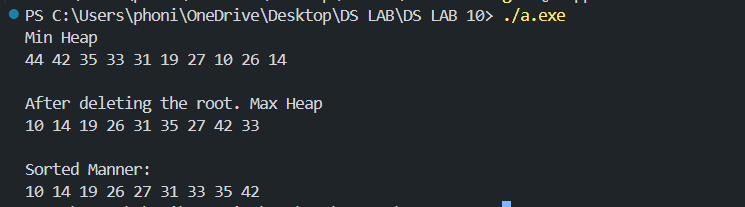
    while(!tmp.isEmpty()){

        cout<<tmp.remove(0)<<" ";

    }

    return 0;

}



TASK 03

#include <iostream>

#include <string>

#include <cstdlib>

#include <ctime>

using namespace std;

void swap(int& a,int& b){

    int tmp=a;

    a=b;

    b=tmp;

}

template<typename T1,typename T2>

class Pair{

    public:

        T1 first;

        T2 second;

    Pair(): first(T1()),second(T2()){};

    Pair(T1 \_first, T2 \_second): first(\_first),second(\_second){};

};

class priorityQueue {

private:

    Pair<string,int> arr[300];

    int pos;

    void heapify\_up(int index) {

        if (index<=0) return;

        int parent=(index-1)/2;

        if (arr[parent].second<arr[index].second) {

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

            heapify\_up(parent);

        }

    }

    void heapify\_down(int index) {

        int leftChild=2\*index+1;

        int rightChild=2\*index+2;

        int minIndex=index;

        if (leftChild<=pos && arr[leftChild].second>arr[minIndex].second) minIndex = leftChild;

        if (rightChild<=pos && arr[rightChild].second>arr[minIndex].second) minIndex = rightChild;

        if (minIndex != index) {

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

            heapify\_down(minIndex);

        }

    }

public:

    priorityQueue() : pos(-1) {}

    int getSize() {

        return pos+1;

    }

    void push(Pair<string,int> node) {

        if (pos+1>=300) return;

        arr[++pos]=node;

        heapify\_up(pos);

    }

    Pair<string,int> pop(){

        if (pos<0) return Pair<string,int>("",0);

        Pair<string,int> top=arr[0];

        arr[0]=arr[pos--];

        heapify\_down(0);

        return top;

    }

    bool isEmpty(){

        return pos<0;

    }

};

int main(){

    priorityQueue queue;

    srand(static\_cast<unsigned>(time(0)));

    for (int i = 0; i < 10; i++) {

        queue.push(Pair<string, int>("t"+to\_string(i), rand()%20));

    }

    while(!queue.isEmpty()){

        Pair<string,int> tmp=queue.pop();

        cout<<tmp.first<<": "<<tmp.second<<endl;

    }

}

A computer screen shot of a computer

Description automatically generated