#include <iostream>

#include <utility>

#include<string>

#define TOP 0;

using namespace std;

enum priori

{

high = 3,

medium = 2,

low = 1

};

class ERheap

{

private:

struct Node

{

int priority;

string name;

string description;

//bool operator < (Node& value);

//heap[length] > heap[p]

//would need comperison operator

};

Node\* heap;

int capacity;

int bottom;

void ReHeapUp(int root, int length);

void ReHeapDown();

public:

ERheap(int s);

~ERheap();

void enqueue(string name, string desc, int pri);

void dequeue(string& name);

bool isFull();

};

ERheap::ERheap(int s)

{

capacity = s;

bottom = 0;

heap = new Node[s];

}

ERheap::~ERheap()

{

delete[] heap;

}

void ERheap::enqueue(string name, string desc, int pri)

{ // sorting the list in descending order. This means the highest priority item will always be at element 0.

//The algorithm for enqueueing a value is going to almost the exact same as the insert algorithm for a sorted list,

//with one slight modification

if (! isFull())

{

bottom = bottom + 1;

heap[bottom - 1].priority = pri;

heap[bottom - 1].name = name;

heap[bottom - 1].description = desc;

ReHeapUp(0, bottom - 1);

/\*

result = 1;

i = 0;

while (i < capacity && pri < heap[i].priority)

{

i = i + 1;

}

j = bottom;

while (j > i)

{

heap[j] = heap[j - 1];

j = j - 1;

}

heap[i].priority = pri;

heap[i].name = name;

heap[i].description = desc;

bottom = bottom + 1;

\*/

}

return;

}

void ERheap::dequeue(string& name)

{ // the list of priorities is sorted in descending order.

//This means, the algorithm basically acts as remove with 0 passed to it.

if (bottom > 0)

{

name = heap[0].name;

heap[0] = heap[bottom - 1];

bottom--;

ReHeapDown();

}

}

void ERheap::ReHeapUp(int root, int length)

{

//int i;

int p;

//i = bottom;

/\*

while (i != 0) //?????

{

p = (i - 1) / 2;

if (heap[i].priority > heap[p].priority)

{

swap(heap[p], heap[i]);

}

i = p;

}\*/

if (length > root)

{

p = (length - 1) / 2;

//heap[length] > heap[p]

//would need comperison operator

if (heap[length].priority > heap[p].priority) //heap[length] > heap[p]

{

swap(heap[p], heap[length]);

ReHeapUp(root, p);

}

}

}

bool ERheap::isFull()

{

if (bottom >= capacity)

{

return true;

}

else

return false;

}

void ERheap::ReHeapDown()

{

int p;

p = 0;

int lc;

int rc;

int bc;

while (p < bottom)

{

lc = 2 \* p + 1;

rc = 2 \* p + 2;

if (lc <= bottom)

{

if (lc == bottom)

{

bc = lc;

}

else

{

if (heap[lc].priority > heap[rc].priority)

{

bc = lc;

}

else

{

bc = rc;

}

}

}

else

return;

if (heap[p].priority < heap[bc].priority)

{

swap(heap[p], heap[bc]);

p = bc;

}

else

{

return;

}

}

}

int main()

{

static const string CHOICES = "ANQ";

ERheap\* menuListPtr = new ERheap(5);

string menuChoiceStr = " ";

char menuChoiceChr = ' ';

////string name, string desc, int pri

string inputPri, inputName, inputDesc, inputQty;

while (menuChoiceChr != 'Q')

{

cout << "(A)dd Patient" << endl;

cout << "(N)ext Patient" << endl;

cout << "(Q)uit" << endl;

menuChoiceStr = " ";

menuChoiceChr = ' ';

while (CHOICES.find(menuChoiceChr) == string::npos)

{

cout << ":] ";

getline(cin, menuChoiceStr);

menuChoiceChr = menuChoiceStr[0];

menuChoiceChr = toupper(menuChoiceChr);

}

switch (menuChoiceChr)

{

case 'A': ////////////// Add

cout << " Enter patient's name: ";

getline(cin, inputName); //read the whole line inlcude space

cout << " Enter complaint: ";

getline(cin, inputDesc);

cout << " Enter priority: ";

getline(cin, inputPri);

menuListPtr->enqueue(inputName, inputDesc, stoi(inputPri));

break;

case 'N': //////////////// Next

cout << " Next Patient: ";

inputName = "None";

menuListPtr->dequeue(inputName);

cout << inputName << endl;

break;

case 'Q': //////////////// Quit

break;

default:

break;

}

}

delete menuListPtr;

return 0;

}