\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Heap.h 18 20

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

#include <iomanip>

using namespace std;

template<class T>

class BinaryMaxHeap

{

private:

T\* elements;

int length;

int numberOfElements;

public:

BinaryMaxHeap(int length)

{

this->length = length;

numberOfElements = 0;

elements = new T[length+1];

for(int i=0; i<length+1; i++)

elements[i] = -1;

}

BinaryMaxHeap(const BinaryMaxHeap<T>& heap)

{

length = heap.length;

numberOfElements = heap.numberOfElements;

elements = new T[length + 1];

for(int i=0; i<length+1; i++)

elements[i] = heap.elements[i];

}

~BinaryMaxHeap()

{

if(elements!=NULL)

delete[] elements;

}

void insert(T element)

{

if(numberOfElements==length)

throw "The heap is full!";

int i = numberOfElements+1;

while(i>1 && elements[i/2]<element)

{

elements[i] = elements[i/2];

i /= 2;

}

elements[i] = element;

numberOfElements++;

}

void print()

{

for(int i=0; i<length+1; i++)

cout << setw(3) << right << elements[i];

cout << endl;

}

***18. Projektovati klasu za rad sa statičkim binarnim maxheap-om i implementirati metod za spajanje dva maxheap-a u jedan.***

BinaryMaxHeap<T> mergeHeaps(BinaryMaxHeap<T>& h)

{

BinaryMaxHeap<T> result(length+h.length);

for(int i=1; i<numberOfElements+1; i++)

result.insert(elements[i]);

for(int i=1; i<h.numberOfElements+1; i++)

result.insert(h.elements[i]);

return result;

}

***20. Projektovati klasu za rad sa statičkim binarnim maxheap-om i implementirati metod za njegovu konverziju u minheap.***

void convertToMinHeap()

{

BinaryMaxHeap<T> heap(\*this);

delete[] elements;

elements = new T[length+1];

for(int i=0; i<length+1; i++)

elements[i] = -1;

numberOfElements = 0;

for(int i=1; i<heap.numberOfElements+1; i++)

this->insertToMinHeap(heap.elements[i]);

}

void insertToMinHeap(T element)

{

if(numberOfElements==length)

throw "The heap is full!";

int i = numberOfElements+1;

while(i>1 && elements[i/2]>element)

{

elements[i] = elements[i/2];

i /= 2;

}

elements[i] = element;

numberOfElements++;

}

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

main.cpp 18 20

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include "Heap.h"

int main()

{

BinaryMaxHeap<int> heap1(8);

BinaryMaxHeap<int>\* heap2 = new BinaryMaxHeap<int>(8);

heap1.insert(12);

heap1.insert(28);

heap1.insert(32);

heap1.insert(45);

cout << "Inorder prvog hipa: ";

heap1.print();

heap2->insert(35);

heap2->insert(14);

heap2->insert(17);

heap2->insert(50);

cout << "Inorder drugog hipa: ";

heap2->print();

cout << "Inorder spojenih hipova: ";

heap1.mergeHeaps(\*heap2).print();

delete heap2;

cout << "Inorder MinHip konverzije: ";

heap1.convertToMinHeap();

heap1.print();

return 0;

}