L05. ARBORI HEAP

Header.h

```
#pragma once

#include <iostream>
using namespace std;

#define DIM 100

void InsertHeap(int* v, int& n, int a);
int DeleteRoot(int*v, int& n);

void BuildHeap(int* v, int n);
void Retrograd(int* v, int n, int i);

void HeapGen(int*v, int n);

void Sort(int*v, int n);
```

Functii.cpp

```
#include "Header.h"

void InsertHeap(int* v, int& n, int a)
{
	int child, father, aux;
	n++;
	v[n] = a;
	child = n;
	father = (n-1) / 2;
	//cout << "\n[INSERT]";
	while (father >= 0)
	{
		//cout << "\nchild= " << child << " father= " << father;
		//cout << "\n v[child]=" << v[child] << " v[father]=" <<
v[father] << endl;
		if (v[child] > v[father])
		{
			//cout << "\n [SCHIMBA] child= " << child << "
father= " << father;
			//cout << "\n v[child]=" << v[child] << "
v[father]=" << v[father] << endl;
			aux = v[child];
```

```
                  v[child] = v[father];
                  v[father] = aux;
                  child = father;
                  if (child)
                        father = (child - 1) / 2;
                  else
                        father = -1;
            }
            else
                  father = -1;
      }
}

int DeleteRoot(int*v, int& n)
{
      int a = v[0];
      int father, child, aux;
      v[0] = v[n];
      n--;
      father = 0;
      child = 1;
      //cout << "\n[DELETE]";
      while (child <= n)
      {
            //cout << "\n father=" << father << " child=" << child;
            //cout << "\n v[child]=" << v[child] << " v[father]=" <<
v[father] << endl;
            if (child + 1 < n && v[child + 1] > v[child])
                  child++;
            if (v[child] > v[father])
            {
                  //cout << "\n [SCHIMBA] child= " << child << "
father= " << father;
                  //cout << "\n v[child]=" << v[child] << "
v[father]=" << v[father] << endl;
                  aux = v[child];
                  v[child] = v[father];
                  v[father] = aux;
                  father = child;
                  child = father * 2 - 1;
            }
            else
                  child = n+1;
      }
      return a;
}

void BuildHeap(int* v, int n)
{
      for (int i = (n+1) / 2;i >= 0;i--)
      {
```

```
            Retrograd(v, n, i);
            //cout << "\n n=" << n << " i=" << i;
      }
}

void Retrograd(int* v, int n, int i)
{
      int father = i;
      int child = 2 * i;
      int aux;
      while (child <= n)
      {
            if (child + 1 <= n && v[child] < v[child + 1])
                  child++;
            if (v[child] > v[father])
            {
                  aux = v[child];
                  v[child] = v[father];
                  v[father] = aux;

                  father = child;
                  child = father * 2;
            }
            else
                  child = n + 1;
      }
}

void HeapGen(int*v, int n)
{
      int aux;
      for (int i = 1;i <=n+1;i++)
      {
            aux = v[i];
            InsertHeap(v, --i, aux);
            //cout << "\n Insert " << aux << " poz " << i;
      }
}

void Sort(int*v, int n)
{
      int aux;
      while (n!=-1)
      {
            aux = DeleteRoot(v, n);
            v[n+1] = aux;
            //cout << "\n n=" << n << " aux=" << aux;
      }
}
```

Main.cpp

```
#include "Header.h"

int main()
{

	cout << endl << "Introducere si stergere radacina:\n";

	int*v, n, a;
	v = new int[DIM];
	n = -1;

	cout << endl << "Elementele de introdus: [30, 15, 25, 10, 12,
20, 18, 3, 5, 7, 8, 11]";
	cout << "\n a = ";
	cin >> a;
	while (a)
	{
		InsertHeap(v, n, a);
		cout << "\n a = ";
		cin >> a;
	}

	cout << endl << "Heap dupa inserare elemente: ";
	for (int i = 0;i <= n;i++)
		cout << v[i] << " ";

	cout << endl << "Inserare 28:";
	a = 28;
	InsertHeap(v, n, a);

	cout << endl << "Heap dupa inserare 28: ";
	for (int i = 0;i <= n;i++)
		cout << v[i] << " ";
	cout << endl;

	a = DeleteRoot(v, n);
	cout << endl << "Radacina: " << a;
	cout << endl << "Heap dupa eliminarea radacinii: ";
	for (int i = 0;i <= n;i++)
		cout << v[i] << " ";
	cout << endl;


	cout << "\n Sortare & Build Heap:";

	int*w1, *w2, m, i;
	w1 = new int[DIM];
	w2 = new int[DIM];

	cout << "\n Vectorul intial:";
```

```
	i = 0;
	cout << "\nw1["<<i<<"]=";
	cin >> w1[i];
	w2[i] = w1[i];
	for (i = 1;w1[i-1]!=0;i++)
	{
		cout << "\nw[" << i << "]=";
		cin >> w1[i];
		w2[i] = w1[i];
	}
	m = i-2;

	cout << endl << "Vectorul initial w1: ";
	for (i = 0;i <= m;i++)
		cout << w1[i] << " ";
	cout << endl;

	BuildHeap(w1, m);

	cout << endl << "Heap w1: ";
	for (i = 0;i <= m;i++)
		cout << w1[i] << " ";
	cout << endl;

	cout << endl << "Vectorul initial w2: ";
	for (i = 0;i <= m;i++)
		cout << w2[i] << " ";
	cout << endl;

	HeapGen(w2, m);

	cout << endl << "Heap w2: ";
	for (i = 0;i <= m;i++)
		cout << w2[i] << " ";
	cout << endl;

	Sort(w1, m);
	cout << endl << "Sorted w1: ";
	for (i = 0;i <= m;i++)
		cout << w1[i] << " ";
	cout << endl;

	Sort(w2, m);
	cout << endl << "Sorted w2: ";
	for (i = 0;i <= m;i++)
		cout << w2[i] << " ";
	cout << endl;

	return 0;
}
```