**Лабораторная работа №12. Бинарные кучи**

**Вариант 10**

Условие:

В проект добавить следующие функции: удаление минимального **ExtractMin**;вывод значений элементов на экран **Scan**; объединение **Union** двух куч в одну. Написать программу с использованием меню.

Код:

Главная функция:

#include "stdafx.h"

#include "Heap.h"

#include <iostream>

#include <fstream>

using namespace std;

heap::CMP cmpAAA(void\* a1, void\* a2) //Функция сравнения

{

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

heap::CMP rc = heap::EQUAL;

if (A1->x > A2->x) rc = heap::GREAT;

else if (A2->x > A1->x) rc = heap::LESS;

return rc;

#undef A2

#undef A1

}

bool BuildHeap(char \*FileName, heap::Heap& h) // Построение кучи из файла

{

bool rc = true;

int n;

ifstream inFile;

inFile.open(FileName, std::ios::out);

if (!inFile)

{

cout << "Невозможно открыть файл" << endl; exit(1);

}

while (inFile >> n)

{

int \*a = new int;

\*a = n;

h.Insert((void\*)a);

}

inFile.close(); return rc;

}

int getnum() {

int k;

std::cout << "Введите число" << std::endl;

std::cin >> k;

return k;

}

void SaveHeap(heap::Heap &h, char \*FileName) // Функция записи в файл

{

ofstream outFile(FileName, ios\_base::out | ios\_base::trunc);

if (!outFile)

{

cout << "Ошибка открытия выходного файла" << std::endl;

return;

}

int \*a = new int;

for (int u = 0, y = 0; u < h.Size; u++)

{

a = (int\*)h.Storage[u];

outFile << \*a;

outFile << endl;

}

outFile.close();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

int k;

int choise;

heap::Heap h1 = heap::Create(30, cmpAAA);

heap::Heap h2 = heap::Create(30, cmpAAA);

do

{

cout << "1 - вывод кучи на экран" << endl;

cout << "2 - добавить элемент в первую кучу" << endl;

cout << "3 - добавить элемент в вторую кучу" << endl;

cout << "4 - очистить кучу" << endl;

cout << "5 - сохранить в файл" << endl;

cout << "6 - загрузить из файла" << endl;

cout << "7 - удалить минимальный элемент" << endl;

cout << "8 - объединение двух куч" << endl;

cout << "9 - удалить максимальный элемент" << endl;

cout << "0 - выход" << endl;

cout << "Введите : "; cin >> choise;

switch (choise)

{

case 0: exit(0);

case 1: {

int k;

cout << "Для вывода 1-й кучи - 1" << endl;

cout << "Для вывода 2-й кучи - 2" << endl;

cout << "Введите: "; cin >> k;

cout << endl;

if (k == 1) {

h1.Scan(0);

}

else if (k == 2) {

h2.Scan(0);

}

cout << endl;

} break;

case 2: {AAA \*a = new AAA; cout << "Введите ключ" << endl; cin >> k;

a->x = k; h1.Insert(a); break; }

case 3: {

AAA \*a = new AAA; cout << "Введите ключ" << endl; cin >> k;

a->x = k; h2.Insert(a); break;

}

case 4: {

int k;

cout << "Для очистки первой кучи введите - 1" << endl;

cout << "Для очистки второй кучи введите - 2" << endl;

cout << "Введите: "; cin >> k;

if (k == 1) {

h1.DeleteHeap();

}

else if (k == 2) {

h2.DeleteHeap();

}

} break;

case 5: SaveHeap(h1, "b.txt"); break;

case 6: h1.DeleteHeap(); BuildHeap( "a.txt", h1); break;

case 7: h1.ExtractMin(); break;

case 8: h1.Union(h2); break;

case 9: h1.ExtractMax(); break;

}

} while (choise != 0);

return 0;

}

Heap.cpp

#include "stdafx.h"

#include "Heap.h"

#include <iostream>

#include <iomanip>

void AAA::Print()

{

std::cout << x;

}

int AAA::GetPriority() const

{

return x;

}

namespace heap {

//Создаение кучи

Heap Create(int maxsize, CMP(\*f)(void\*, void\*)) {

return \*(new Heap(maxsize, f));

}

void Heap::Insert(void\* x)

{

int i;

if (!isFull()) {

Storage[i = ++Size - 1] = (void\*)x; //следующий адрес равен указателю на переданные данные

while (i > 0 && isLess(Storage[Parent(i)], Storage[i]))

{

Swap(Parent(i), i);

i = Parent(i);

}

}

}

int Heap::Left(int cur)

{

return (2 \* cur + 1 >= Size) ? -1 : (2 \* cur + 1); //формулы вычисления потомков и родителя

}

int Heap::Right(int cur) {

return (2 \* cur + 2 >= Size) ? -1 : (2 \* cur + 2);

}

int Heap::Parent(int cur) {

return (cur + 1) / 2 - 1;

}

void Heap::Swap(int i, int j)//перестановка двух элементов массива указателей

{

void\* buf = Storage[i];

Storage[i] = Storage[j];

Storage[j] = buf;

}

void Heap::Heapify(int cur)

{

int left\_cur = Left(cur), right\_cur = Right(cur), irl = cur;

if (left\_cur > 0)

{

if (isGreat(Storage[left\_cur], Storage[cur])) irl = left\_cur;

if (right\_cur > 0 && isGreat(Storage[right\_cur], Storage[irl])) irl = right\_cur;

if (irl != cur) {

Swap(cur, irl);

Heapify(irl); }

}

}

void\* Heap::ExtractMax()//удаление максимального

{

void\* rc = NULL;

if (!isEmpty())

{

rc = Storage[0];

Storage[0] = Storage[Size - 1];

Size--;

Heapify(0);

}

return rc;

}

void\* Heap::ExtractMin() //удаление минимального

{

int key = 99999, index = 0;

for (int u = 0; u < Size; u++)

{

if (((AAA\*)Storage[u])->x < key)

{

key=((AAA\*)Storage[u])->x;

index = u;

}

}

void\* buf = NULL;

if (!isEmpty()) {

buf = Storage[Size];

Storage[index] = Storage[Size - 1];

Size--;

Heapify((index+1)/2+1);

}

return buf;

}

void\* Heap::ExtractI() { //удаление по номеру

void\* buf = NULL;

int num;

std::cout << "Введите ключ: "; std::cin >> num;

if (!isEmpty()) {

buf = Storage[num];

Storage[num] = Storage[Size - 1];

Size--;

Heapify((num+1)/2+1);

}

return buf;

}

void Heap::Scan(int i) const //Вывод значений элементов на экран

{

int probel = 20; std::cout << '\n';

if (Size == 0) std::cout << "Куча пуста";

for (int u = 0, y = 0; u < Size; u++)

{

std::cout << std::setw(probel + 10) << std::setfill(' ');

((AAA\*)Storage[u])->Print();

if (u == y) { std::cout << '\n'; if (y == 0) y = 2; else y += y \* 2; }

probel /= 2;

}

std::cout << '\n';

}

void Heap::DeleteHeap()

{

if (!isEmpty())

{

Size = 0;

this->~Heap(); //очистка кучи через деструктор

}

}

void Heap::Union(heap::Heap h) {

int i = 0;

while (h.Size) {

Insert(h.ExtractMax());

}

h.DeleteHeap();

}

}

Heap.h

#pragma once

struct AAA { // информация об элементах кучи

int x; // приоритет

void Print(); //вывод

int GetPriority() const;

};

namespace heap

{

enum CMP

{

LESS = -1, EQUAL = 0, GREAT = 1

};

struct Heap // структура кучи

{

int Size;

int MaxSize;

void\*\* Storage; //ссылка на хранилище

CMP(\*Compare)(void\*, void\*); //сравнение двух элементов

Heap(int maxsize, CMP(\*f)(void\*, void\*))//конструктор кучи

{

Size = 0; //Размер кучи

Storage = new void \*[MaxSize = maxsize];

Compare = f;

};

Heap(int maxsize, CMP(\*f)(void\*, void\*), void\* x[])//конструктор кучи

{

Size = 0;

Storage = x;

MaxSize = maxsize;

Compare = f;

};

int Left(int ix);

int Right(int ix);

int Parent(int ix);

bool isFull() const {

return (Size >= MaxSize);

};// проверка на переполнение

bool isEmpty() const {

return (Size <= 0);

};//проверка на пустоту

bool isLess(void\* x1, void \* x2) const {

return Compare(x1, x2) == LESS;

};

bool isGreat(void\* x1, void\* x2) const {

return Compare(x1, x2) == GREAT;

};

bool isEqual(void\* x1, void\* x2) const {

return Compare(x1, x2) == EQUAL;

};

void Swap(int i, int j);//Перестановка 2 элементов

void Heapify(int ix);

void Insert(void\* x); //ввод элемента

void\* ExtractMax();//удаление максимального

void\* ExtractMin();//удаление минимального

void\* ExtractI(); //удаление элемента по номеру

void DeleteHeap(); //очистка кучи

void Scan(int i) const;// вывод кучи

void outMas();// выводит элементы кучи по порядку

void Union(heap::Heap h);

};

Heap Create(int maxsize, CMP(\*f)(void\*, void\*));

};

Скрин

