Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 12

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные кучи»

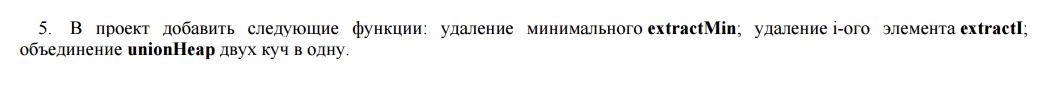
Выполнил:

Студент 1 курса 6 группы

Романов Игорь Вячеславович

Преподаватель: асс. Андронова М.В.

2024, Минск



В проект также добавлены 3 дополнительные функции:

1. Поиск минимального элемента в куче
2. Поиск максимального элемента в куче
3. Очистка кучи

**Heap.h**

#pragma once

using namespace std;

//Элементы кучи

struct AAA {

int x;

void print() const; //печать значений

};

namespace heap {

//типы сравения

enum CMP {

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

};

struct Heap {

int size;

int maxSize;

void\*\* storage; // данные

CMP(\*compare)(void\*, void\*);

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

size = 0; //начальный размер

storage = new void\* [maxSize = maxsize]; //хранилище элементов

compare = f;

};

int left(int ix) const; //Получение левого потомка

int right(int ix) const; //Получение правого потомка

int parent(int ix); //Получение предка

bool isFull() const {

return (size >= maxSize);

};

bool isEmpty() const {

return (size <= 0);

};

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);

void heapify(int ix);

void insert(void\* x);

void\* extractMax();

void\* extractMin(); //прототип функции удаления минимального элемента

void\* extractI(int i); // прототип функции удаления i-ого элемента

void unionHeap(Heap& other); // прототип функции объединения двух куч

void getMin(); // прототип функции вывода минимального значения в куче

void getMax(); // прототип функции вывода максимального значения в куче

void scan(int i) const;

void clear(); //фукнция очистки кучи

};

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

};

**Heap.cpp**

#include "Heap.h"

#include <iostream>

#include <iomanip>

//Печать элементов из кучи

void AAA::print() const {

cout << x;

}

namespace heap {

Heap create(int maxsize, CMP(\*f)(void\*, void\*)) { // Создание кучи с заданным максимальным размером и функцией сравнения

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

}

int Heap::left(int ix) const { //Получение индекса левого потомка

return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1);

}

int Heap::right(int ix) const { //Получение индекса правого потомка

return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2);

}

int Heap::parent(int ix) { //Получение индекса родительского элемента

return (ix + 1) / 2 - 1;

}

void Heap::swap(int i, int j) { // Метод для обмена элементов

void\* buf = storage[i];

storage[i] = storage[j];

storage[j] = buf;

}

void Heap::heapify(int ix) { // Метод для преобразования кучи в кучу

int l = left(ix), r = right(ix), irl = ix;

if (l > 0) {

if (isGreat(storage[l], storage[ix])) {

irl = l;

}

if (r > 0 && isGreat(storage[r], storage[irl])) {

irl = r;

}

if (irl != ix) {

swap(ix, irl);

heapify(irl);

}

}

}

void Heap::insert(void\* x) { // Метод для вставки элемента в кучу

int i;

if (!isFull()) {

storage[i = ++size - 1] = x;

while (i > 0 && isLess(storage[parent(i)], storage[i])) {

swap(parent(i), i);

i = parent(i);

}

}

}

// Метод для извлечения максимального элемента из кучи

void\* Heap::extractMax() {

void\* rc = nullptr;

if (!isEmpty()) {

rc = storage[0];

storage[0] = storage[size - 1];

size--;

heapify(0);

} return rc;

}

//Вывод значений элементов на экран

void Heap::scan(int i) const {

int probel = 20; // Начальное количество пробелов перед каждым значением

cout << '\n'; // Переход на новую строку перед выводом значений

// Проверяем, пуста ли куча

if (size == 0) {

cout << "Куча пуста"; // Если куча пуста, выводим соответствующее сообщение

}

// Проходим по всем элементам кучи для вывода значений

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

cout << setw(probel + 10) << setfill(' '); // Устанавливаем отступ перед значением

// Выводим значение элемента, приведенное к типу AAA

((AAA\*)storage[u])->print();

// Проверяем, нужно ли переходить на новую строку для вывода следующего уровня

if (u == y) {

cout << '\n'; // Переходим на новую строку

if (y == 0) {

y = 2; // Если это первый уровень, устанавливаем следующий уровень на 2

}

else {

y += y \* 2; // Иначе увеличиваем следующий уровень в два раза

}

}

// Уменьшаем количество пробелов перед следующим значением в два раза

probel /= 2;

}

cout << '\n'; // Переход на новую строку после вывода всех значений

}

// Удаление минимального элемента

void\* Heap::extractMin() {

void\* rc = nullptr;

if (!isEmpty()) {

rc = storage[size - 1];

size--;

heapify(0);

}

return rc;

}

// Удаление i-ого элемента

void\* Heap::extractI(int i) {

void\* rc = nullptr;

if (!isEmpty() && i >= 0 && i < size) {

rc = storage[i];

storage[i] = storage[size - 1];

size--;

heapify(0);

}

return rc;

}

// Объединение двух куч

void Heap::unionHeap(Heap& other) {

for (int i = 0; i < other.size; i++) {

insert(other.storage[i]);

}

other.size = 0; //очищение дополнительной кучи

}

// Вывод минимального значения в куче

void Heap::getMin() {

if (!isEmpty()) {

// Инициализируем минимальное значение как значение первого элемента

void\* minVal = storage[0];

// Проходим по всем элементам кучи, начиная со второго

for (int i = 1; i < size; ++i) {

// Если текущий элемент меньше минимального, обновляем минимальное значение

if (compare(storage[i], minVal) == LESS) {

minVal = storage[i];

}

}

cout << "Минимальный элемент в куче: ";

((AAA\*)minVal)->print();

cout << endl;

}

else {

cout << "Куча пуста" << endl;

}

}

// Вывод максимального значения в куче

void Heap::getMax() {

if (!isEmpty()) {

cout << "Максимальный элемент в куче: ";

((AAA\*)storage[0])->print();

cout << std::endl;

}

else {

cout << "Куча пуста" << endl;

}

}

// Функция для очистки кучи (удаление всех элементов)

void Heap::clear() {

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

delete (AAA\*)storage[i]; // Освобождаем память для каждого элемента

}

size = 0; // Устанавливаем размер кучи в 0

cout << "Куча очищена!" << endl;

}

}

**Main.cpp**

#include "Heap.h"

#include <iostream>

using namespace std;

// Принимает два указателя на void, которые приводятся к указателям на AAA

heap::CMP cmpAAA(void\* a1, void\* a2) {

// Приведение указателей к типу AAA

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

heap::CMP rc = heap::EQUAL; // Инициализация результата сравнения как EQUAL

// Сравниваем значения x элементов типа AAA

if (A1->x > A2->x) {

rc = heap::GREAT; // Если значение x первого элемента больше, результат - GREAT

}

else if (A2->x > A1->x) {

rc = heap::LESS; // Если значение x второго элемента больше, результат - LESS

}

return rc;

// Удаляем макросы для предотвращения конфликтов с другими частями кода

#undef A2

#undef A1

}

//главная фукнция

int main() {

setlocale(LC\_ALL, "rus");

int k, choice;

heap::Heap h1 = heap::create(30, cmpAAA); //Создание кучи

for (;;) {

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

cout << "2 - добавить элемент" << endl;

cout << "3 - удалить максимальный элемент" << endl;

cout << "4 - удалить минимальный элемент" << endl;

cout << "5 - удалить i-ый элемент" << endl;

cout << "6 - объединить две кучи" << endl;

cout << "7 - вывод минимального значения в куче" << endl;

cout << "8 - вывод максимального значения в куче" << endl;

cout << "9 - очистить кучу" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl; cin >> choice;

switch (choice) {

case 0: {

exit(0);

}

case 1: {

h1.scan(0);

break;

}

case 2: {

AAA\* a = new AAA;

cout << "введите ключ" << endl; cin >> k;

a->x = k;

h1.insert(a);

break;

}

case 3: {

h1.extractMax();

break;

}

case 4: { // Добавляем обработчик для удаления минимального элемента

h1.extractMin();

break;

}

case 5: { // Добавляем обработчик для удаления i-ого элемента

int i;

cout << "введите индекс элемента для удаления" << endl; cin >> i;

h1.extractI(i);

break;

}

case 6: { // Добавляем обработчик для объединения двух куч

heap::Heap h2 = heap::create(20, cmpAAA); // Создание второй кучи

// Добавляем элементы во вторую кучу h2, если нужно

h1.unionHeap(h2);

break;

}

case 7: { // Добавляем обработчик для вывода минимального значения в куче

h1.getMin();

break;

}

case 8: { // Добавляем обработчик для вывода максимального значения в куче

h1.getMax();

break;

}

case 9: {

h1.clear(); // Очистить кучу

break;

}

default: {

cout << endl << "Введена неверная команда!" << endl;

}

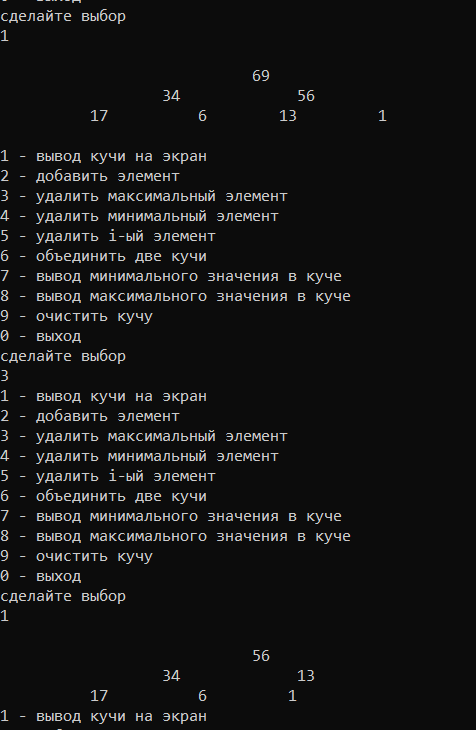
}

}

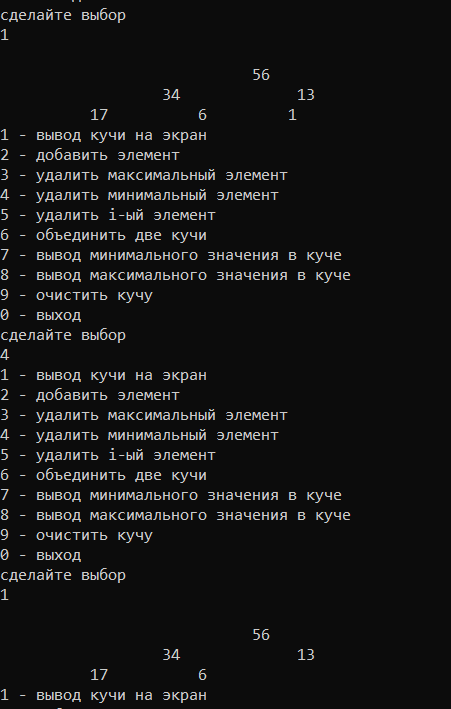
return 0;

}

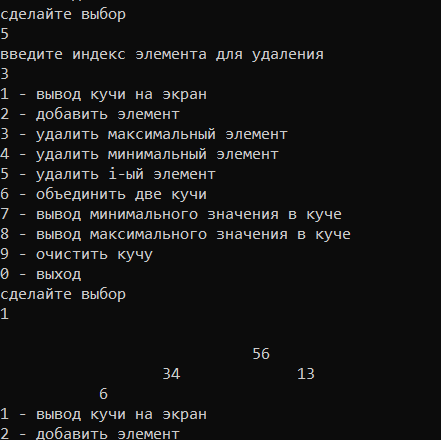
Удаление максимального элемента:



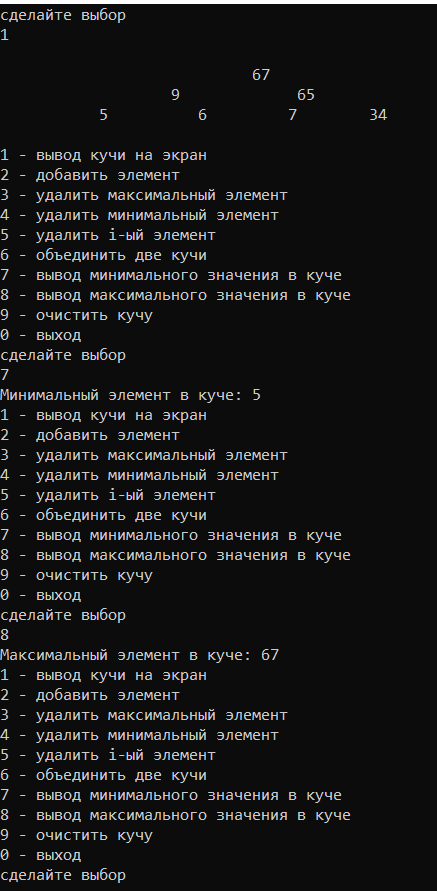
Удаление минимального элемента:



Удаление элемента по заданному значению:



Вывод минимального и максимального значения в куче:



Очистка в куче:

