Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 12

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные кучи»

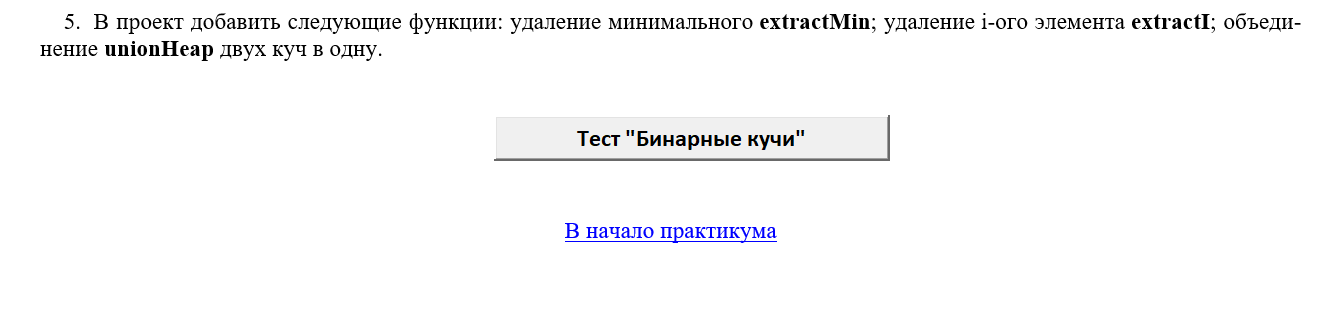
 Выполнила:

Студентка1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: асс. Андронова М.В.

2023, Минск



Main.cpp

#include "Heap.h"

#include <iostream>

#include <Windows.h>

using namespace std;

// Функция для сравнения двух объектов типа AAA в куче.

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

{

// Приведение указателей к типу AAA.

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

// Инициализация результата сравнения как EQUAL (равенство).

heap::CMP rc = heap::EQUAL;

// Сравнение значений 'x' объектов AAA.

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

rc = heap::GREAT; // Если A1.x > A2.x, результат GREAT (больше).

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

rc = heap::LESS; // Если A2.x > A1.x, результат LESS (меньше).

// Возврат результата сравнения.

return rc;

// Удаление макросов для предотвращения возможных конфликтов.

#undef A2

#undef A1

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Объявление переменных.

int k, choice, index;

// Создание экземпляра кучи h1 с емкостью 30 и использованием функции сравнения cmpAAA.

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

// Бесконечный цикл для взаимодействия с пользователем.

for (;;)

{

// Отображение вариантов меню.

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 элемент по индексу" << endl;

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

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl;

// Чтение выбора пользователя.

cin >> choice;

// Инструкции switch в зависимости от выбора пользователя.

switch (choice)

{

case 0: // Выход из программы.

exit(0);

case 1: // Отобразить содержимое кучи.

h1.scan(0);

break;

case 2: // Добавить элемент в кучу.

{

// Динамическое выделение памяти для объекта AAA.

AAA\* a = new AAA;

cout << "введите ключ" << endl;

cin >> k;

a->x = k;

// Вставить элемент в кучу.

h1.insert(a);

}

break;

case 3: // Удалить максимальный элемент из кучи.

h1.extractMax();

break;

case 4: // Удалить минимальный элемент из кучи.

h1.extractMin();

break;

case 5: // Удалить элемент по индексу из кучи.

cout << "введите индекс элемента для удаления" << endl;

cin >> index;

h1.extractI(index);

break;

case 6: // Объединить две кучи.

{

cout << "Создание второй кучи для объединения..." << endl;

heap::Heap h2 = heap::create(2, cmpAAA);

for (size\_t i = 0; i < 2; i++)

{

// Динамическое выделение памяти для объекта AAA.

AAA\* a = new AAA;

cout << "введите ключ" << endl;

cin >> k;

a->x = k;

// Вставить элемент в кучу.

h2.insert(a);

}

// Можно добавить элементы в h2 по мере необходимости.

cout << "Объединение двух куч" << endl;

h1.unionHeap(h2);

}

break;

default: // Вывести сообщение о неправильной команде.

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

}

}

return 0;

}

Heap.h

#pragma once

struct AAA

{

int x;

void print();

};

namespace heap

{

enum CMP

{

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

};

struct Heap

{

int size; // Текущий размер кучи

int maxSize; // Максимальный размер кучи

void\*\* storage; // Массив указателей на данные

CMP(\*compare)(void\*, void\*); // Указатель на функцию сравнения

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

{

size = 0;

storage = new void\* [maxSize = maxsize]; // Выделение памяти под массив хранения данных

compare = f; // Инициализация указателя на функцию сравнения

};

int left(int ix); // Возвращает индекс левого потомка для элемента с индексом ix.

int right(int ix); // Возвращает индекс правого потомка для элемента с индексом ix.

int parent(int ix); // Возвращает индекс родительского элемента для элемента с индексом ix.

bool isFull() const

{

return (size >= maxSize); // Проверка, заполнена ли куча

};

bool isEmpty() const

{

return (size <= 0); // Проверка, пуста ли куча

};

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

{

return compare(x1, x2) == LESS; // Проверка, является ли x1 меньшим чем x2

};

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

{

return compare(x1, x2) == GREAT; // Проверка, является ли x1 большим чем x2

};

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

{

return compare(x1, x2) == EQUAL; // Проверка, равны ли x1 и x2

};

void swap(int i, int j); // Меняет местами элементы с индексами i и j

void heapify(int ix); // Поддерживает свойство кучи, начиная с элемента с индексом ix

void insert(void\* x); // Вставляет новый элемент в кучу

void\* extractMax(); // Извлекает и возвращает максимальный элемент кучи

void scan(int i) const; // Сканирует элемент с индексом i

void\* extractMin(); // Извлекает и возвращает минимальный элемент кучи

void extractI(int i); // Извлекает элемент с индексом i

void unionHeap(Heap& h2); // Объединяет две кучи

};

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

};

Heap.cpp

#include "Heap.h"

#include <iostream>

#include <iomanip>

// Функция print() для вывода значения элемента AAA.

void AAA::print()

{

std::cout << x; // Вывод значения x элемента AAA.

}

namespace heap

{

// Функция create() для создания кучи с заданным максимальным размером и функцией сравнения.

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

{

return \*(new Heap(maxsize, f)); // Возвращает новую кучу с заданными параметрами.

}

// Функция left() для вычисления индекса левого потомка элемента в куче.

int Heap::left(int ix)

{

return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1); // Возвращает индекс левого потомка.

}

// Функция right() для вычисления индекса правого потомка элемента в куче.

int Heap::right(int ix)

{

return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2); // Возвращает индекс правого потомка.

}

// Функция parent() для вычисления индекса родительского элемента в куче.

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1; // Возвращает индекс родительского элемента.

}

// Функция swap() для обмена элементов в куче.

void Heap::swap(int i, int j)

{

void\* buf = storage[i]; // Временная переменная для хранения элемента.

storage[i] = storage[j]; // Замена элемента на позиции i элементом на позиции j.

storage[j] = buf; // Обновление элемента на позиции j из временной переменной.

}

// Функция heapify() для преобразования поддерева с корнем в узле ix в кучу.

void Heap::heapify(int ix)

{

int l = left(ix), r = right(ix), irl = ix; // Вычисление индексов левого и правого потомков.

if (l > 0) // Проверка, существует ли левый потомок.

{

if (isGreat(storage[l], storage[ix])) irl = l; // Если левый потомок больше текущего узла, обновляем индекс максимума.

if (r > 0 && isGreat(storage[r], storage[irl])) irl = r; // Если правый потомок больше текущего максимума, обновляем индекс максимума.

if (irl != ix) // Если индекс максимума отличается от текущего индекса, выполняем обмен элементов и рекурсивно вызываем heapify().

{

swap(ix, irl); // Обмен текущего узла с узлом максимума.

heapify(irl); // Вызов heapify для обновленного узла максимума.

}

}

}

// Функция insert() для вставки нового элемента в кучу.

void Heap::insert(void\* x)

{

int i;

if (!isFull()) // Проверка на заполненность кучи.

{

storage[i = size++] = x; // Вставка нового элемента в кучу и увеличение размера кучи.

while (i > 0 && isLess(storage[parent(i)], storage[i])) // Поддержание свойства кучи после вставки нового элемента.

{

swap(parent(i), i); // Обмен элементов, если свойство кучи нарушено.

i = parent(i); // Переход к родительскому узлу.

}

}

}

// Функция extractMax() для извлечения максимального элемента из кучи.

void\* Heap::extractMax()

{

void\* rc = nullptr; // Инициализация возвращаемого значения.

if (!isEmpty()) // Проверка на пустоту кучи.

{

rc = storage[0]; // Сохранение максимального элемента.

storage[0] = storage[size - 1]; // Перемещение последнего элемента в корень кучи.

size--; // Уменьшение размера кучи.

heapify(0); // Поддержание свойства кучи.

}

return rc; // Возвращение извлеченного максимального элемента.

}

// Функция scan() для вывода значений элементов кучи на экран.

void Heap::scan(int i) const

{

int probel = 20; // Начальное значение для форматирования вывода.

std::cout << '\n'; // Переход на новую строку.

if (size == 0)

std::cout << "Куча пустая"; // Вывод сообщения о пустой куче.

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

{

std::cout << std::setw(probel + 10) << std::setfill(' '); // Форматированный вывод значений элементов.

((AAA\*)storage[u])->print(); // Вызов функции печати элемента AAA.

if (u == y)

{

std::cout << '\n'; // Переход на новую строку.

if (y == 0)

y = 2;

else

y += y \* 2; // Увеличение шага для форматирования вывода.

}

probel /= 2; // Уменьшение значения для форматирования вывода.

}

std::cout << '\n'; // Переход на новую строку.

}

// Функция extractMin() для извлечения минимального элемента из кучи.

void\* Heap::extractMin()

{

void\* rc = nullptr; // Инициализация возвращаемого значения.

if (!isEmpty()) // Проверка на пустоту кучи.

{

int minIndex = 0; // Индекс минимального элемента, начинаем с 0.

// Поиск минимального элемента среди дочерних элементов.

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

if (isLess(storage[i], storage[minIndex])) {

minIndex = i; // Обновление индекса минимального элемента, если найден более маленький.

}

}

rc = storage[minIndex]; // Сохранение минимального элемента перед удалением.

storage[minIndex] = storage[size - 1]; // Замена минимального элемента последним элементом в куче.

size--; // Уменьшение размера кучи.

heapify(minIndex); // Поддержание свойства кучи после удаления минимального элемента.

}

return rc; // Возвращение извлеченного минимального элемента.

}

// Функция extractI() для удаления элемента по заданному индексу из кучи.

void Heap::extractI(int i)

{

if (i < 0 || i >= size) // Проверка на корректность индекса.

{

std::cerr << "Index out of bounds." << std::endl; // Вывод сообщения об ошибке.

return; // Возврат из функции.

}

// Обмен элемента, который нужно удалить, с последним элементом, а затем поддержание свойства кучи.

swap(i, size - 1);

size--; // Уменьшение размера кучи.

heapify(i); // Поддержание свойства кучи.

}

// Функция unionHeap() для объединения двух куч.

void Heap::unionHeap(Heap& h2)

{

for (int i = 0; i < h2.size; ++i) // Перебор элементов второй кучи.

{

insert(h2.storage[i]); // Вставка элементов второй кучи в текущую кучу.

}

h2.size = 0; // Очистка второй кучи после объединения.

}

};

