

Дап. Заданні:

Тры функцыі:

// Функцыя для атрымання максімальнага элемента ў кучы без яго выдалення.

void\* Heap::getMax()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

// Вяртаем каранёвы элемент кучы (максімальны элемент).

return storage[0];

else

// Калі куча пустая, вяртаем nullptr.

return nullptr;

}

// Функцыя для атрымання мінімальнага элемента ў кучы без яго выдалення.

void\* Heap::getMin()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

{

int minIndex = 0;

// Шукаем мінімальны элемент сярод усіх элементаў кучы.

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

{

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

minIndex = i;

}

// Вяртаем знойдзены мінімальны элемент.

return storage[minIndex];

}

else

{

// Калі куча пустая, вяртаем nullptr.

return nullptr;

}

}

// Функцыя для пошуку элемента ў кучы, які роўны суме максімальнага і мінімальнага элементаў.

void\* Heap::findSumElement()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

{

// Атрымліваем максімальны і мінімальны элементы кучы.

void\* maxElement = getMax();

void\* minElement = getMin();

// Правяраем, ці знойдзены максімальны і мінімальны элементы.

if (maxElement != nullptr && minElement != nullptr)

{

// Вылічаем суму максімальнага і мінімальнага элементаў.

int sum = ((AAA\*)maxElement)->x + ((AAA\*)minElement)->x;

// Перабіраем элементы кучы для пошуку элемента, роўнага суме.

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

{

// Калі бягучы элемент роўны суме, вяртаем яго.

if (((AAA\*)storage[i])->x == sum)

return storage[i];

}

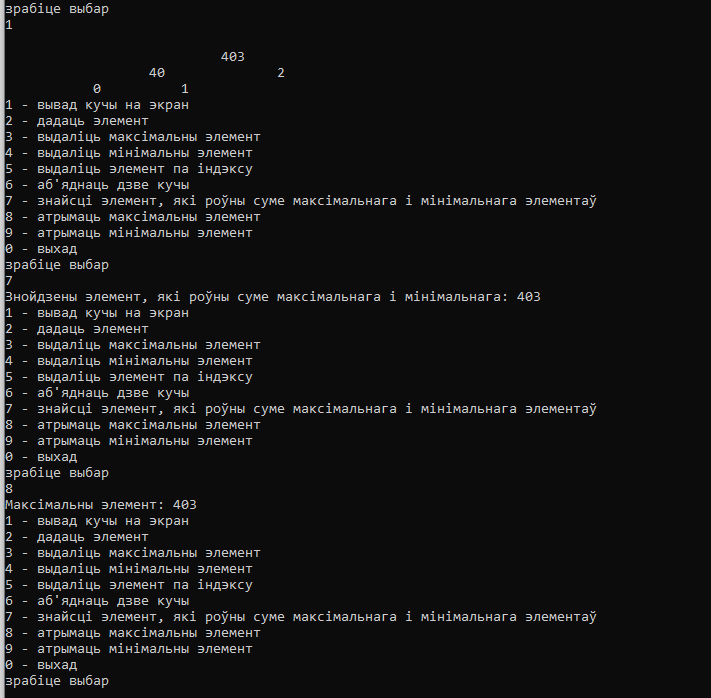
}

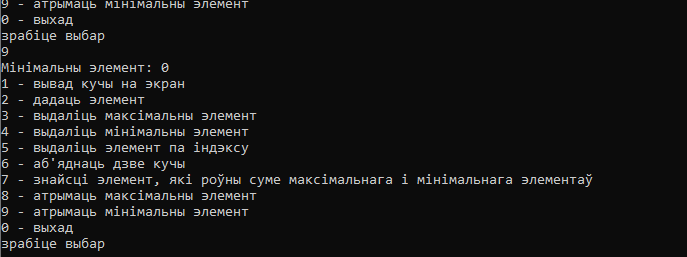
}

// Калі не ўдалося знайсці элемент, роўны суме максімальнага і мінімальнага, вяртаем nullptr.

return nullptr;

}





**Увесь код:**

Main.cpp:

#include "Heap.h"

#include <iostream>

#include <Windows.h>

using namespace std;

// Функцыя для параўнання двух аб'ектаў тыпу AAA у кучы.

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

{

// Прывядзенне паказальнікаў да тыпу AAA.

#define A1 ((AAA\*)a1)

#define A2 ((AAA\*)a2)

// Ініцыялізацыя выніку параўнання як EQUAL (роўнасць).

heap::CMP rc = heap::EQUAL;

// Параўнанне значэнняў 'x' аб'ектаў AAA.

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

rc = heap::GREAT; // Калі A1.x > A2.x, вынік GREAT (больш).

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

rc = heap::LESS; // Калі A2.x > A1.x, вынік LESS (менш).

// Зварот выніку параўнання.

return rc;

// Выдаленне макрасаў для прадухілення магчымых канфліктаў.

#undef A2

#undef A1

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Аб'ява зменных.

int k, choice, index;

// Стварэнне асобніка кучы h1 з ёмістасцю 30 і выкарыстаннем функцыі параўнання cmpAAA.

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

// Бясконцы цыкл для ўзаемадзеяння з карыстальнікам.

for (;;)

{

// Адлюстраванне варыянтаў меню.

cout << "1 - вывад кучы на экран" << endl;

cout << "2 - дадаць элемент" << endl;

cout << "3 - выдаліць максімальны элемент" << endl;

cout << "4 - выдаліць мінімальны элемент" << endl;

cout << "5 - выдаліць элемент па індэксу" << endl;

cout << "6 - аб'яднаць дзве кучы" << endl;

cout << "7 - знайсці элемент, які роўны суме максімальнага і мінімальнага элементаў" << endl; // Новы пункт мяню

cout << "8 - атрымаць максімальны элемент" << endl; // Новы пункт мяню

cout << "9 - атрымаць мінімальны элемент" << endl; // Новы пункт мяню

cout << "0 - выхад" << endl;

cout << "зрабіце выбар" << endl;

// Чытанне выбару карыстальніка.

cin >> choice;

// Інструкцыі switch у залежнасці ад выбару карыстальніка.

switch (choice)

{

case 0: // Выхад з праграмы.

exit(0);

case 1: // Адлюстраваць змесціва кучы.

h1.scan(0);

break;

case 2: // Дадаць элемент у кучу.

{

// Дынамічнае вылучэнне памяці для аб'екта AAA.

AAA\* a = new AAA;

cout << "увядзіце ключ" << endl;

cin >> k;

a->x = k;

// Уставіць элемент у кучу.

h1.insert(a);

}

break;

case 3: // Выдаліць максімальны элемент з кучы.

h1.extractMax();

break;

case 4: // Выдаліць мінімальны элемент з кучы.

h1.extractMin();

break;

case 5: // Выдаліць элемент па індэксе з кучы.

cout << "увядзіце індэкс элемента для выдалення" << endl;

cin >> index;

h1.extractI(index);

break;

case 6: // Аб'яднаць дзве кучы.

{

cout << "Стварэнне другой кучы для аб'яднання..." << endl;

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

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

{

// Дынамічнае вылучэнне памяці для аб'екта AAA.

AAA\* a = new AAA;

cout << "увядзіце ключ" << endl;

cin >> k;

a->x = k;

// Уставіць элемент у кучу.

h2.insert(a);

}

// Можна дадаць элементы ў h2 па меры неабходнасці.

cout << "Аб'яднанне двух куч" << endl;

h1.unionHeap(h2);

}

break;

case 7: // Пошук элемента, роўнага суме максімальнага і мінімальнага элементаў

{

void\* sumElement = h1.findSumElement();

if (sumElement != nullptr)

{

cout << "Знойдзены элемент, які роўны суме максімальнага і мінімальнага: ";

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

cout << endl;

}

else

{

cout << "Не атрымалася знайсці элемент, які роўны суме максімальнага і мінімальнага." << endl;

}

}

break;

case 8: // Атрыманне максімальнага элемента з кучы

{

void\* maxElement = h1.getMax();

if (maxElement != nullptr)

{

cout << "Максімальны элемент: ";

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

cout << endl;

}

else

{

cout << "Куча пустая." << endl;

}

}

break;

case 9: // Атрыманне мінімальнага элемента з кучы

{

void\* minElement = h1.getMin();

if (minElement != nullptr)

{

cout << "Мінімальны элемент: ";

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

cout << endl;

}

else

{

cout << "Куча пустая." << endl;

}

}

break;

default: // Вывесці паведамленне аб няслушнай камандзе.

cout << endl << "Уведзена няслушная каманда!" << 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; // Бягучы памер кучы

int maxSize; // Максімальны памер кучы

void\*\* storage; // Масіў паказальнікаў на дадзеныя

CMP(\*compare)(void\*, void\*); // Паказальнік на функцыю параўнання

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

{

size = 0;

storage = new void\* [maxSize = maxsize]; // Вылучэнне памяці пад масіў захоўвання дадзеных

compare = f; // Ініцыялізацыя паказальніка на функцыю параўнання

};

int left(int ix); // Вяртае азначнік левага нашчадка для элемента з азначнікам ix.

int right(int ix); // Вяртае азначнік правага нашчадка для элемента з азначнікам ix.

int parent(int ix); // Вяртае азначнік бацькоўскага элемента для элемента з азначнікам ix.

bool isFull() const

{

return (size >= maxSize); // Праверка, ці запоўненая куча

};

bool isEmpty() const

{

return (size <= 0); // Праверка, ці пустая куча

};

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

{

return compare(x1, x2) == LESS; // Праверка, ці з'яўляецца x1 меншым за x2

};

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

{

return compare(x1, x2) == GREAT; // Праверка, ці з'яўляецца x1 большым за x2

};

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

{

return compare(x1, x2) == EQUAL; // Праверка, ці роўныя x1 і x2

};

void swap(int i, int j); // Змяняе месцамі элементы з індэксамі i і j

void heapify(int ix); // Падтрымлівае ўласцівасць кучы, пачынаючы з элемента з індэксам ix

void insert(void\* x); // Устаўляе новы элемент у кучу

void\* extractMax(); // Здабывае і вяртае максімальны элемент кучы

void scan(int i) const; // Скануе элемент з індэксам i

void\* extractMin(); // Здабывае і вяртае мінімальны элемент кучы

void extractI(int i); // Дастае элемент з індэксам i

void unionHeap(Heap& h2); // Аб'ядноўвае дзве кучы

void\* getMax();

void\* getMin();

void\* findSumElement();

};

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

};

Heap.cpp:

#include "Heap.h"

#include <iostream>

#include <iomanip>

// Функцыя print() для вываду значэння элемента AAA.

void AAA::print()

{

std::cout << x; // Вывад значэння x элемента AAA.

}

namespace heap

{

// Функцыя для атрымання максімальнага элемента ў кучы без яго выдалення.

void\* Heap::getMax()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

// Вяртаем каранёвы элемент кучы (максімальны элемент).

return storage[0];

else

// Калі куча пустая, вяртаем nullptr.

return nullptr;

}

// Функцыя для атрымання мінімальнага элемента ў кучы без яго выдалення.

void\* Heap::getMin()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

{

int minIndex = 0;

// Шукаем мінімальны элемент сярод усіх элементаў кучы.

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

{

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

minIndex = i;

}

// Вяртаем знойдзены мінімальны элемент.

return storage[minIndex];

}

else

{

// Калі куча пустая, вяртаем nullptr.

return nullptr;

}

}

// Функцыя для пошуку элемента ў кучы, які роўны суме максімальнага і мінімальнага элементаў.

void\* Heap::findSumElement()

{

// Правяраем, ці не пустая куча.

if (!isEmpty())

{

// Атрымліваем максімальны і мінімальны элементы кучы.

void\* maxElement = getMax();

void\* minElement = getMin();

// Правяраем, ці знойдзены максімальны і мінімальны элементы.

if (maxElement != nullptr && minElement != nullptr)

{

// Вылічаем суму максімальнага і мінімальнага элементаў.

int sum = ((AAA\*)maxElement)->x + ((AAA\*)minElement)->x;

// Перабіраем элементы кучы для пошуку элемента, роўнага суме.

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

{

// Калі бягучы элемент роўны суме, вяртаем яго.

if (((AAA\*)storage[i])->x == sum)

return storage[i];

}

}

}

// Калі не ўдалося знайсці элемент, роўны суме максімальнага і мінімальнага, вяртаем nullptr.

return nullptr;

}

// Функцыя create() для стварэння кучы з зададзеным максімальным памерам і функцыяй параўнання.

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

{

return \*(new Heap(maxsize, f)); // Вяртае новую кучу з зададзенымі параметрамі.

}

// Функцыя left() для вылічэння індэкса левага нашчадка элемента ў кучы.

int Heap::left(int ix)

{

return (2 \* ix + 1 >= size) ? -1 : (2 \* ix + 1); // Вяртае індэкс левага нашчадка.

}

// Функцыя right() для вылічэння індэкса правага нашчадка элемента ў кучы.

int Heap::right(int ix)

{

return (2 \* ix + 2 >= size) ? -1 : (2 \* ix + 2); // Вяртае індэкс правага нашчадка.

}

// Функцыя parent() для вылічэння індэкса бацькоўскага элемента ў кучы.

int Heap::parent(int ix)

{

return (ix + 1) / 2 - 1; // Вяртае індэкс бацькоўскага элемента.

}

// Функцыя swap() для абмену элементаў у кучы.

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

{

void\* buf = storage[i]; // Часавая пераменная для захоўвання элемента.

storage[i] = storage[j]; // Замена элемента на пазіцыі i элементам на пазіцыі j.

storage[j] = buf; // Аднаўленне элемента на пазіцыі j з часовай зменнай.

}

// Функцыя heapify() для пераўтварэння паддрэва з коранем у вузле ix у кучу.

void Heap::heapify(int ix)

{

int l = left(ix), r = right(ix), irl = ix; // Вылічэнне індэксаў левага і правага нашчадкаў.

if (l > 0) // Праверка, ці існуе левы нашчадак.

{

if (isGreat(storage[l], storage[ix])) irl = l; // Калі левы нашчадак больш за бягучы вузл, абнаўляем індэкс максімуму.

if (r > 0 && isGreat(storage[r], storage[irl])) irl = r; // Калі правы нашчадак больш за бягучы максімум, абнаўляем індэкс максімуму.

if (irl != ix) // Калі азначнік максімуму адрозніваецца ад бягучага азначніка, выконваем абмен элементаў і рэкурсіўна выкліканы heapify().

{

swap(ix, irl); // Абмен бягучага вузла з вузлом максімуму.

heapify(irl); // Выклік heapify для абноўленага вузла максімуму.

}

}

}

// Функцыя insert() для ўстаўкі новага элемента ў кучу.

void Heap::insert(void\* x)

{

int i;

if (!isFull()) // Праверка на запоўненасць кучы.

{

storage[i = size++] = x; // Устаўка новага элемента ў кучу і павелічэнне памеру кучы.

while (i > 0 && isLess(storage[parent(i)], storage[i])) // Падтрыманне ўласцівасці кучы пасля ўстаўкі новага элемента.

{

swap(parent(i), i); // Абмен элементаў, калі ўласцівасць кучы парушана.

i = parent(i); // Пераход да бацькоўскага вузла.

}

}

}

// Функцыя extractMax() для вымання максімальнага элемента з кучы.

void\* Heap::extractMax()

{

void\* rc = nullptr; // Ініцыялізацыя якое вяртаецца значэння.

if (!isEmpty()) // Праверка на пустэчу кучы.

{

rc = storage[0]; // Захаванне максімальнага элемента.

storage[0] = storage[size - 1]; // Перамяшчэнне апошняга элемента ў корань кучы.

size--; // Памяншэнне памеру кучы.

heapify(0); // Падтрыманне ўласцівасці кучы.

}

return rc; // Вяртанне вынятага максімальнага элемента.

}

// Функцыя scan() для вываду значэнняў элементаў кучы на экран.

void Heap::scan(int i) const

{

int probel = 20; // Пачатковае значэнне для фарматавання вываду.

std::cout << '\n'; // Пераход на новы радок.

if (size == 0)

std::cout << "Куча пустая"; // Выснова паведамлення аб пустой кучы.

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

{

std::cout << std::setw(probel + 10) << std::setfill(' '); // Фарматаваная выснова значэнняў элементаў.

((AAA\*)storage[u])->print(); // Выклік функцыі друку элемента AAA.

if (u == y)

{

std::cout << '\n'; // Пераход на новы радок.

if (y == 0)

y = 2;

else

y += y \* 2; // Павелічэнне кроку для фарматавання вываду.

}

probel /= 2; // Памяншэнне значэння для фарматавання вываду.

}

std::cout << '\n'; // Пераход на новы радок.

}

// 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.

}

// Функцыя extractI() для выдалення элемента па зададзеным індэксе з кучы.

void Heap::extractI(int i)

{

if (i < 0 || i >= size) // Праверка на карэктнасць індэкса.

{

std::cerr << "Index out of bounds." << std::endl; // Выснова паведамлення пра памылку.

return; // Зварот з функцыі.

}

// Абмен элемента, які трэба выдаліць, з апошнім элементам, а затым падтрыманне ўласцівасці кучы.

swap(i, size - 1);

size--; // Памяншэнне памеру кучы.

heapify(i); // Падтрыманне ўласцівасці кучы.

}

// Функцыя unionHeap() для аб'яднання двух куч.

void Heap::unionHeap(Heap& h2)

{

for (int i = 0; i < h2.size; ++i) // Перабор элементаў другой кучы.

{

insert(h2.storage[i]); // Устаўка элементаў другой кучы ў бягучую кучу.

}

h2.size = 0; // Ачыстка другой кучы пасля аб'яднання.

}

};