МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра системного проектування

**ЗВІТ**про виконання лабораторної роботи №3

«Дослідження структур даних пріоритетна черга та купа»

з дисципліни «Алгоритми та структури даних»

Виконав:

cтудент І курсу

групи ДА-02

Гринчишин Д.Б.

Перевірили:

..

Київ – 2020

**Мета**. Ознайомитись і дослідити структуру даних бінарна купа. Набути навичок реалізації абстрактної структури даних черга з пріоритетом на основі бінарної купи мовою програмування С++. Ознайомитись з механізмом перевантаження операторів та дослідити особливості сортування купою (пірамідального сортування). Порівняти власну реалізацію пріоритетної черги з готовим бібліотечним рішенням STL.

**Варіант завдань – 2**

**Код:**

#include <iostream>

#include <ctime>

#include <cmath>

#include <string>

#include <cstdlib>

#include <cstring>

#include <queue>

#include <cstdlib>

#include <algorithm>

#include <iomanip>

using namespace std;

struct Data{

    int hp;

    int damage;

    int skill;

    Data(){

        hp = rand() % 100;

        damage = rand() % 100;

        skill = rand() % 100;

    }

};

bool operator<(Data e1, Data e2){ return e1.skill <= e2.skill; }

bool operator>(Data e1, Data e2){ return e1.skill >= e2.skill; }

bool operator==(Data e1, Data e2){ return e1.skill == e2.skill; } //&& e1.hp == e2.hp && e1.damage == e2.damage; }

void operator<<(ostream & os, Data e){

    cout << e.damage << ' ' << e.hp << ' ' << e.skill << ' ';

}

template<typename T> class PriorityQueue{

    public :

        int typesize = sizeof(T);

        int Size = 0;

        int max\_size = 1;

        T\* array = new T[max\_size];

        T\* sort(T\* arr, int len){

            max\_size = len + 10;

            array = new T[max\_size];

            for (size\_t i = 0; i < len; i++) push(arr[i]);

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

            return array;

        }

        void push(T elem){

            if (Size == max\_size){

                max\_size \*= 2;

                T\* temp\_array = new T[max\_size];

                memcpy(temp\_array, array, Size \* typesize);

                destructor();

                array = temp\_array;

            }

            array[Size] = elem;

            siftUp(Size);

            Size++;

        }

        void siftUp(int index){

            for (int parent = getParent(index); array[index] > array[parent] && index > 0; index = parent, parent = getParent(index))

                swap(array[parent], array[index]);

        }

        void siftDown(int parent){

            int temp, left = getLeftChild(parent), right = getRightChild(parent);

            while(left < Size){

                temp = 0;

                if(array[left] > array[parent]) temp = left;

                if (right < Size) {

                    if(array[right] > array[left]) temp = right;

                }

                if(temp != 0){

                    swap(array[temp], array[parent]);

                    parent = temp;

                }

                else break;

                left = getLeftChild(parent), right = getRightChild(parent);

            }

        }

        int getParent(int index){ return index == 0 ? 0 : index % 2 == 0 ? (index - 2) / 2 : (index - 1) / 2; }

        int getLeftChild(int index) { return index \* 2 + 1; }

        int getRightChild(int index) { return index \* 2 + 2; }

        void print(int m = 1000){

            if(Size == 0) cout << "no elem" << endl;

            else{

                for (size\_t i = 0; i < Size && i < m; i++) cout << array[i] << endl;

            }

            cout << endl;

        }

        T top(){ return array[0]; }

        void pop(){

            swap(array[0], array[Size - 1]);

            Size--;

            siftDown(0);

        }

        bool empty(){ return Size <= 0; }

        int size(){ return Size; }

        void destructor(){delete [] array;}

};

template <typename T>

float testPriorityQueueSpeed(T&& priorityQueue)

{

    const int iters = 100000;

    clock\_t timeStart = clock();

    for (int i = 0; i < iters; i++)

    {

        int insertDataAmount = rand() % 6 + 5;

        for (int j = 0; j < insertDataAmount; j++)

        {

            priorityQueue.push(Data());

        }

        priorityQueue.top();

        priorityQueue.pop();

    }

    clock\_t timeEnd = clock();

    float time = (float(timeEnd - timeStart)) / CLOCKS\_PER\_SEC;

    return time;

}

bool testPriorityQueue()

{

    srand(time(NULL));

    const int iters = 20000;

    PriorityQueue<Data> myPriorQueue;

    priority\_queue<Data> stlPriorQueue;

    bool isDataEqual = true;

    for (int i = 0; i < iters; i++)

    {

        int insertDataAmount = rand() % 6 + 5;

        for (int j = 0; j < insertDataAmount; j++)

        {

            Data randData = Data();

            myPriorQueue.push(randData);

            stlPriorQueue.push(randData);

        }

        if (!(myPriorQueue.top() == stlPriorQueue.top()))

        {

            isDataEqual = false;

            cerr << "Comparing failed on iteration " << i << endl << endl;

            break;

        }

        int removeDataAmount = rand() % insertDataAmount;

        for (int j = 0; j < removeDataAmount; j++)

        {

            myPriorQueue.pop();

            stlPriorQueue.pop();

        }

    }

    int myQueueSize = myPriorQueue.size();

    int stlQueueSize = stlPriorQueue.size();

    float stlTime = testPriorityQueueSpeed<priority\_queue<Data>>(priority\_queue<Data>());

    float myTime = testPriorityQueueSpeed<PriorityQueue<Data>>(PriorityQueue<Data>());

    cout << "My PriorityQueue:" << endl;

    cout << "Time: " << myTime << ", size: " << myQueueSize << endl;

    cout << "STL priority\_queue:" << endl;

    cout << "Time: " << stlTime << ", size: " << stlQueueSize << endl << endl;

    if (isDataEqual && myQueueSize == stlQueueSize)

    {

        cout << "The lab is completed" << endl << endl;

        return true;

    }

    cerr << ":(" << endl << endl;

    return false;

}

int main(int argc, char\*\* argv){

    srand(time(0));

    int mode = 0;

    switch (mode) {

        case 0 : {

            bool echo = 0;

            int N;

            cout << "enter number of elements : ";

            cin >> N;

            int\* unsorted = new int[N];

            for (size\_t i = 0; i < N; i++) { unsorted[i] = rand() % 30; if (echo) cout << unsorted[i] << ' '; }

            if (echo) cout << endl;

            clock\_t stl = clock();

            sort(unsorted, unsorted + N);

            clock\_t end\_stl = clock();

            if (echo) for (size\_t i = 0; i < N; i++) cout << unsorted[i] << ' ';

            if (echo) cout << endl;

            clock\_t my = clock();

            PriorityQueue<int> sortingHeap;

            int\* sorted = sortingHeap.sort(unsorted, N);

            clock\_t end\_my = clock();

            sortingHeap.destructor();

            delete [] sorted;

            delete [] unsorted;

            if (echo) for (size\_t i = 0; i < N; i++) cout << sorted[i] << ' ';

            if (echo) cout << endl;

            cout << "heap sort time : " << (double)(end\_my - my) / CLOCKS\_PER\_SEC << endl;

            cout << "stl sort time : " << setprecision(13) << (double)(end\_stl - stl) / CLOCKS\_PER\_SEC << endl;

            break;

        }

        case 1 :{

            testPriorityQueue();

            break;

        }

    }

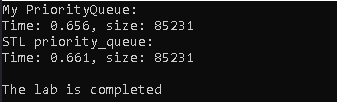
    getchar();

    getchar();

    return 0;

}

**Результат виконання програми:**



**Висновок:**

В ході виконання лабораторної роботи було реалізовано структуру даних бінарна купа. Було розроблена структура даних черга з пріоритетом на основі бінарної купи. Було реалізовано перевантаження операторів >, <, == для структури даних мого варіанту. Моя реалізація була порівняна з STL реалізацією і показала незначно менший час виконання алгоритму. Було реалізовано алгоритм пірамідального сортування на основі раніше реалізованої бінарної купи, яке показує в середньому в 2 раза гірші результати в плані часу виконання в порівнянні з алгоритмом std::sort().