МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 1**

*по дисциплине «Алгоритмы и структуры данных»*

Выполнил студен группы Пибд-12

Ляхов Т.И.

Проверил доцент кафедры

«Информационные системы»

Гуськов. Г. Ю.

Ульяновск, 2024

**Задачи**:

1.Реализовать АТД(Очередь) на структуре данных (односвязаный список) заданной по варианту, для объектов из предметной области(лекарства для пациента).

2. Реализовать сортировку пузырьком и быструю сортировку на массиве

**Описание используемых алгоритмов и структур данных:**

1. Для первой задачи требовалось реализовать АТД Очередь – подчиняющаяся закону first in first out. Элементы из очереди достаются из начала, новые элементы добавляются в конец. Реализовано это было на односвязаном списке. Отличительной чертой данной структуры данных является что каждый элемент представляет из себя структуру, в которой хранится значение данного элемента и указатель на следующий элемент. Плюсы: быстрое изменение размера списка. Минусы: отсутствует индексация, следовательно долгое время обращения к элементу (по сравнению с массивом).
2. Второй задачей было написать сортировки.

Сортировка пузырьком представляет из себя перебор всех элементов массива. За одну итерацию сравниваются два элемента и при надобности меняются местами. Это происходит пока мы не пройдемся по всем элементам массива.

Быстрая сортировка. В основе логики qsort лежит опорный элемент (pivot) относительно которого мы меняем массива. Все элементы меньше пивота становятся слева от него, элементы больше – справа. От выбора пивота зависит время работы алгоритма на тех или иных последовательностях данных. Для удобства, брал pivot 0ой элемент массива.

**Код**

#include <iostream>

#include <string>

#include <fstream>

#include <random>

#include <time.h>

#include <iomanip>

#pragma comment(linker, "/STACK:16777216")

using namespace std;

ifstream in("ListOfMedicines.txt");

class Medicament {

public:

Medicament(long long price, string name) {

this->price = price;

this->nameMedicament = name;

}

void printMed()

{

cout << "(Название: " << nameMedicament << ", Цена: " << price << ")";

}

private:

long long price;

string nameMedicament;

string getNameMedicament() {

return nameMedicament;

}

long long getPrice() {

return price;

}

};

class IntetfaceToAdt

{

public:

virtual void clear() = 0;

virtual bool isEmpty() = 0;

virtual void push(Medicament\* const medicament) = 0;

virtual void pop() = 0;

virtual Medicament\* front() = 0;

virtual void printAdt() = 0;

virtual Medicament\* getForIndex(int index) = 0;

virtual int deleteForIndex(int index) = 0;

};

class AdtQueueWrapper : public IntetfaceToAdt

{

public:

class Queue

{

public:

Medicament\* data;

Queue\* next;

Queue(Medicament\* data) {

this->data = data;

next = nullptr;

}

};

Queue\* head;

AdtQueueWrapper() {

head = nullptr;

}

int deleteForIndex(int index) {

int cnt = 0;

Queue\* realptr = head;

if (index == 0) { this->pop(); return 1; }

while (cnt < index - 1 || realptr == nullptr)

{

if (realptr == nullptr) {

return 0;

}

++cnt;

realptr = realptr->next;

}

if (realptr->next == nullptr || index < 0) { return 0; }

Queue\* getnextpetr = realptr->next, \* twoxnextpr = realptr->next->next;

delete getnextpetr->data;

delete getnextpetr;

realptr->next = twoxnextpr;

return 1;

}

void clear() override

{

if (isEmpty()) { return; }

Queue\* realPtr = head->next;

while (realPtr != nullptr) {

delete head->data;

delete head;

head = realPtr;

realPtr = realPtr->next;

}

head = nullptr;

}

bool isEmpty() override {

return (head == nullptr);

}

void pop() override {

if (isEmpty()) { return; }

Queue\* realptr = head->next;

delete head->data;

delete head;

head = realptr;

}

Medicament\* front() override {

return (\*head).data;

}

void push(Medicament\* medicament) override {

if (isEmpty())

{

this->head = new Queue(medicament);

return;

}

Queue\* realPtr = head;

while (realPtr->next != nullptr) {

realPtr = realPtr->next;

}

realPtr->next = new Queue(medicament);

}

void printAdt() override {

if (isEmpty())

{

cout << "NULL" << endl;

return;

}

head->data->printMed();

cout << " -> ";

Queue\* realPtr = head->next;

while (realPtr != nullptr) {

realPtr->data->printMed();

cout << " -> ";

realPtr = realPtr->next;

}

cout << "NULL" << endl;

}

Medicament\* getForIndex(int index) {

int cnt = 0;

Queue\* realptr = head;

while (cnt != index || realptr == nullptr)

{

if (realptr == nullptr) {

return nullptr;

}

++cnt;

realptr = realptr->next;

}

return realptr->data;

}

};

Medicament\* readNewObj()

{

long long price;

string name;

in >> name >> price;

return new Medicament(price, name);

}

void takeFirstTask() {

setlocale(LC\_ALL, "ru");

AdtQueueWrapper adtqueue;

cout << adtqueue.isEmpty() << endl;

for (int i = 0; i < 3; i++)

{

adtqueue.push(readNewObj());

adtqueue.printAdt();

}

cout << endl;

adtqueue.getForIndex(2)->printMed();

adtqueue.deleteForIndex(0);

cout << endl;

adtqueue.printAdt();

adtqueue.front()->printMed();

cout << endl;

adtqueue.clear();

adtqueue.printAdt();

for (int i = 0; i < 3; i++)

{

adtqueue.push(readNewObj());

}

adtqueue.printAdt();

}

// Task2

void bubbleSort(int\* arr, int size) {

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

{

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

{

if (arr[i] < arr[j])

{

int bf = arr[j];

arr[j] = arr[i];

arr[i] = bf;

}

}

}

}

bool isSorted(int\* arr, int size) {

for (int i = 0; i < size - 1; i++)

{

if (arr[i] > arr[i + 1])

{

return false;

}

}

return true;

}

int\* myQsortMerge(int\* arrFirst, int\* arrSecond, int sizeFirst, int sizeSecond) {

int\* arr = new int[sizeFirst + sizeSecond];

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

{

arr[i] = arrFirst[i];

}

for (int i = sizeFirst; i < sizeFirst + sizeSecond; i++)

{

arr[i] = arrSecond[i - sizeFirst];

}

return arr;

}

int\* myQsort(int\* arr, int size) {

int\* finalarr = new int[size];

int pivot = arr[0];

int leftInd = 0, rightInd = 0;

if (size == 1 || size == 0) {

return arr;

}

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

{

if (pivot > arr[i])

{

finalarr[leftInd] = arr[i];

++leftInd;

}

else

{

finalarr[size - rightInd - 1] = arr[i];

++rightInd;

}

}

finalarr[leftInd] = pivot;

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

{

arr[i] = finalarr[i];

}

delete[] finalarr;

if (isSorted(arr, size)) {

return arr;

}

if (size == 2)

{

return arr;

}

int\* far = myQsort(arr, leftInd + 1);

int\* sar = myQsort(arr + size - rightInd, rightInd);

return myQsortMerge(far, sar, leftInd + 1, rightInd);

}

void takeSecondTask() {

srand(time(0));

int sizeGL = 100000000;

int\* arr = new int[sizeGL];

double sum = 0;

for (int i = 0; i < 1; i++)

{

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

{

arr[i] = rand() \* rand();

}

clock\_t start = clock();

arr = myQsort(arr, sizeGL);

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

sum += seconds;

}

cout << setprecision(8);

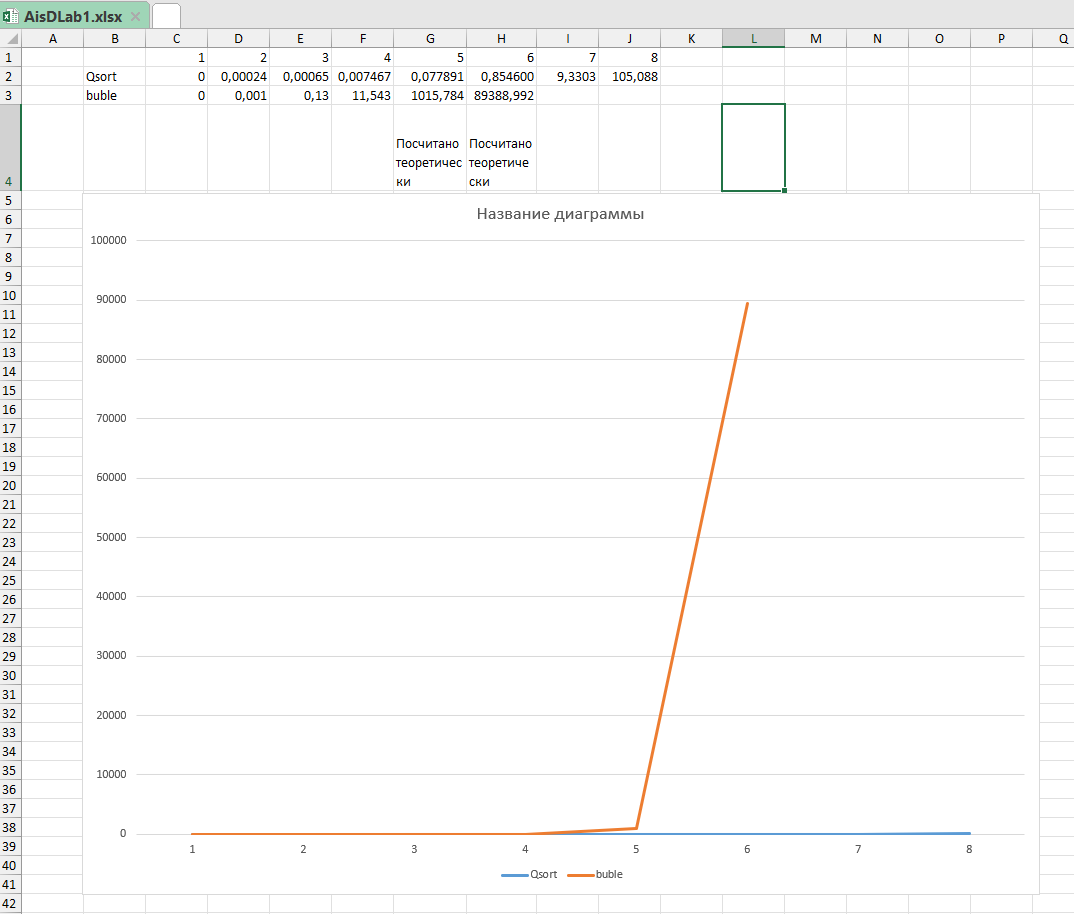
cout << fixed << (double)sum / 1.0 << endl;

}

void main()

{

}

Время работы: (сек)

Выводы:

Реализовал АТД Очередь на односвязанном списке. Улучшил навыки в работе с указателями.

Получилось реализовать сортировку пузырьком и быструю сортировку. (довольно успешно).