Министерство образование Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Информационно-технологический факультет

Кафедра технологий программирования

Лабораторная работа №9

По дисциплина «Основы алгоритмизации и программирования»

Выполнил: Крутилин С.Д.

Группа 19-ИТ-1.

Проверил: Адамовский Е.

Новополоцк

2019 г.

**Сортировка и поиск**

**Вариант 2**

**Цель работы**: изучить быстрые и медленные алгоритмы сортировки. Изучить алгоритмы линейного и бинарного поиска.

**Задание 1.** Организуйте выгрузку данных из Вашей структуры данных в динамический массив.

**Описание:** функция принимает в качестве параметров указатель на массив, в который будет осуществляться выгрузка, и список, из которого будет осуществляться выгрузка. Для того, чтобы пройти по каждому элементу списка создается временная переменная. Заполнение массива будет осуществляться до тех пор, пока эта переменная не станет указывать на конец списка.

void fillArray(Contact\*\* tempArray, doubly\_linked\_list\* L)

{

Contact\* tmp;

tmp = L->start->next\_link;

for (int i = 0; tmp != L->end; i++) {

resize\_z(tempArray, i + 1);

strcpy((\*tempArray)[i].FIO, tmp->FIO);

strcpy((\*tempArray)[i].phone, tmp->phone);

strcpy((\*tempArray)[i].city, tmp->city);

strcpy((\*tempArray)[i].street, tmp->street);

(\*tempArray)[i].house = tmp->house;

(\*tempArray)[i].flat = tmp->flat;

tmp = tmp->next\_link;

}

}

**Задание 2.** Организуйте выгрузку данных из динамического массива в Вашу структуру данных.

**Описание:** В качестве аргумента функция принимает указатель на список, в который осуществляется выгрузка данных, массив, из которого выгружаются данные и размер этого массива. Сама функция использует временную переменную для записи данных из массива, а затем вставки в наш список с помощью ранее написанных функций.

void fillList(doubly\_linked\_list\* L, Contact\* tempArray, int sizeArray)

{

// временный элемент

Contact tempElement;

// пока не конец списка

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

strcpy(tempElement.FIO, tempArray[i].FIO);

strcpy(tempElement.phone, tempArray[i].phone);

strcpy(tempElement.city, tempArray[i].city);

strcpy(tempElement.street, tempArray[i].street);

tempElement.house = tempArray[i].house;

tempElement.flat = tempArray[i].flat;

endPtr(L);

prevPut(L, &tempElement);

}

}

**Задание 3.** Организуйте сортировку элементов массива по любому полю любым алгоритмом.

**Описание:** В качестве алгоритмы я выбрал сортировка шелла. Сортировка Шелла – усовершенствованная версия сортировки вставками. Как показывают опыты, алгоритм вставками куда быстрее работает с почти упорядоченными массивами. Сортировка Шелла учитывает этот фактор и делает почти отсортированный массив искусственно.

Функция принимает указатель на список, создает и заполняет временный массив, который мы будем в дальнейшем сортировать с помощью выбранного алгоритма сортировки. Также создает третью переменную через которую будет осуществляться вставка элемента в нужную позицию. После выполнения всех итераций цикла мы получаем отсортированный массив. Дело остается за малым: заново записать элементы в наш список и очистить память.. Что функция и делает.

void fioAscending(doubly\_linked\_list\* L)

{

Contact\* tempArray = NULL;

fillArray(&tempArray, L);

Contact\* tempElement = NULL;

resize\_z(&tempElement, 1);

int arraySize = numElements(L);

for (int step = arraySize / 2; step > 0; step /= 2) {

for (int i = step; i < arraySize; i++) {

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

if (strcmp(tempArray[j].FIO, tempArray[i].FIO) > 0) {

tempElement[0] = tempArray[j];

tempArray[j] = tempArray[i];

tempArray[i] = tempElement[0];

}

}

}

}

clearList(L);

initialize(L);

fillList(L, tempArray, arraySize);

clear\_z(&tempArray);

clear\_z(&tempElement);

}

void fioDescending(doubly\_linked\_list\* L)

{

Contact\* tempArray = NULL;

fillArray(&tempArray, L);

Contact\* tempElement = NULL;

resize\_z(&tempElement, 1);

int arraySize = numElements(L);

for (int step = arraySize / 2; step > 0; step /= 2) {

for (int i = step; i < arraySize; i++) {

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

if (strcmp(tempArray[j].FIO, tempArray[i].FIO) < 0) {

tempElement[0] = tempArray[j];

tempArray[j] = tempArray[i];

tempArray[i] = tempElement[0];

}

}

}

}

clearList(L);

initialize(L);

fillList(L, tempArray, arraySize);

clear\_z(&tempArray);

clear\_z(&tempElement);

}

**Задание 4.** Организуйте медленный поиск записей в Вашей структуре данных (или быстрый поиск в предварительно выгруженном и отсортированном динамическом массиве) по любому полю.

**Описание:** в меню реализован выбор по какому полю осуществляется поиск. В функции мы просим пользователя ввести запрос. Делаем проверку на правильность ввода. Создаем временный список, в который будем записывать найденные элементы. Создаем дополнительный элемент, который нужен для того, чтобы пробежать по всем элементам списка. Если элемент найдет, то копируем данные из него и вставляем в наш список. Затем предоставляем пользователя возможность просмотреть найденные элементы. Очищаем временный список.

void searchStreet(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите улицу проживания: " << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_street(request) == false);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (strcmp(request, tmp->street) == 0) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

**Задание 5.** Добавьте написанные Вами функции в курсовую работу.

**Описание:** функции поиска по каждому полю были добавлены в курсовую работу в виде модуля search

#include "Search.h"

void searchFIO(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите ФИО: " << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_FIO(request) == false);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (strcmp(request, tmp->FIO) == 0) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

void searchFirstName(doubly\_linked\_list\* L)

{

}

void searchSecondName(doubly\_linked\_list\* L)

{

}

void searchMiddleName(doubly\_linked\_list\* L)

{

}

void searchPhone(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите номер телефона: " << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_phone(request) == false);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (strcmp(request, tmp->phone) == 0) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

void searchCity(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите город проживания: " << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_city(request) == false);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (strcmp(request, tmp->city) == 0) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

void searchStreet(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите улицу проживания: " << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_street(request) == false);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (strcmp(request, tmp->street) == 0) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

void searchHouse(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите номер дома" << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_house(request) == false);

int numRequest = atoi(request);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (numRequest == tmp->house) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

void searchFlat(doubly\_linked\_list\* L)

{

char request[100];

do {

cout << "Введите номер квартиры" << endl;

cin.getline(request, 100);

system("cls");

} while (s\_check\_flat(request) == false);

int numRequest = atoi(request);

int sizeList = numElements(L);

// создание и инициализация списка

doubly\_linked\_list\* tempList = (doubly\_linked\_list\*)malloc(0);

initialize(tempList);

// дополнительный элемент для прохода от начала до конца списка

Contact\* tmp;

tmp = L->start->next\_link; // ссылка на первый элемент

// временный элемент

Contact tempElement;

// пока не конец списка

while (tmp != L->end) {

if (numRequest == tmp->flat) {

strcpy\_s(tempElement.FIO, tmp->FIO);

strcpy\_s(tempElement.phone, tmp->phone);

strcpy\_s(tempElement.city, tmp->city);

strcpy\_s(tempElement.street, tmp->street);

tempElement.house = tmp->house;

tempElement.flat = tmp->flat;

tempElement.next\_link = NULL;

tempElement.prev\_link = NULL;

// перемещение указателя на фиктивный конец

endPtr(tempList);

// вставка в конец списка

prevPut(tempList, &tempElement);

}

tmp = tmp->next\_link;

}

table(tempList);

clearList(tempList);

}

bool s\_check\_FIO(char(&str)[100]) {

size\_t length\_str = strlen(str);

int num\_of\_spaces = 0;

for (unsigned i = 0; i < length\_str; i++) {

if (str[i] == ' ') num\_of\_spaces++;

}

if (num\_of\_spaces == length\_str) {

return false;

}

if (str[0] == ' ' || length\_str > 40) {

return false;

}

else return true;

}

bool s\_check\_phone(char(&str)[100])

{

size\_t length\_str = strlen(str);

if (length\_str != 12) {

return false;

}

for (unsigned i = 0; i < length\_str; i++) {

if (str[i] < '0' || str[i] > '9') {

return false;

}

}

return true;

}

bool s\_check\_city(char(&str)[100])

{

size\_t length\_str = strlen(str);

int num\_of\_spaces = 0;

for (unsigned i = 0; i < length\_str; i++) {

if (str[i] == ' ') num\_of\_spaces++;

}

if (num\_of\_spaces == length\_str) {

return false;

}

if (str[0] == ' ' || length\_str > 20) {

return false;

}

else return true;

}

bool s\_check\_street(char(&str)[100])

{

size\_t length\_str = strlen(str);

int num\_of\_spaces = 0;

for (unsigned i = 0; i < length\_str; i++) {

if (str[i] == ' ') num\_of\_spaces++;

}

if (num\_of\_spaces == length\_str) {

return false;

}

if (str[0] == ' ' || length\_str > 20) {

return false;

}

else return true;

}

bool s\_check\_house(char(&str)[100])

{

if (strlen(str) > 3 || str[0] == ' ' || str[0] == '0') {

return false;

}

for (unsigned i = 0; i < strlen(str); i++) {

if (str[i] < '0' || str[i] > '9') {

return false;

}

}

return true;

}

bool s\_check\_flat(char(&str)[100])

{

if (strlen(str) > 3 || str[0] == ' ' || str[0] == '0') {

return false;

}

for (unsigned i = 0; i < strlen(str); i++) {

if (str[i] < '0' || str[i] > '9') {

return false;

}

}

return true;

}