Министерство Образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

**СТРУКТУРЫ ДАННЫХ**

**Редактирование файлов: удаление, изменение поля.**

**Лабораторная работа № 10, 11.**

Студент:

1курса 2 группы 1 подгруппы

Марушко Тимофей Фёдорович

Проверяющий:

Белодед Николай Иванович

**ОГЛАВЛЕНИЕ**

[**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 3**](#_Toc195089772)

[**1. ФУНКЦИЯ ВЫБОРОЧНОГО УДАЛЕНИЯ 3**](#_Toc195089773)

[**2. ФУНКЦИЯ ПОЛНОГО УДАЛЕНИЯ 6**](#_Toc195089774)

[**РЕШЕНИЕ ОСНОВНЫХ ЗАДАЧ 8**](#_Toc195089775)

Теоретические сведения

1. Функция выборочного удаления

**Функция выборочного удаления** – это функция, позволяющая удалить из файла определенную строку или группу строк, не изменяя содержимое других строк в файле.

Для реализации функции выборочного удаления можно использовать следующий алгоритм:

1. Открыть файл для чтения и создать временный файл для записи.
2. Считать строки из оригинального файла и записывать их во временный файл.
3. При считывании строк проверять, соответствует ли текущая строка той строке, которую нужно удалить. Если да, то не записывать ее во временный файл.
4. Закрыть оба файла и переименовать временный файл в оригинальный файл.

Использование функции выборочного удаления может быть полезным при работе с большими файлами, когда необходимо удалить только часть информации из файла, не затрагивая остальное содержимое.

**Нюансы** при использовании функции выборочного удаления:

1. Изменение индексов элементов. При удалении элемента из массива, индексы всех последующих элементов смещаются на одну позицию влево. Это может привести к ошибкам, если в программе используются жестко закодированные индексы элементов массива. Чтобы избежать этой проблемы, можно использовать итераторы или ссылки на элементы массива.
2. Выделение и освобождение памяти. При удалении элемента из массива, необходимо выделить новый массив, который будет содержать оставшиеся элементы. При этом надо учитывать, что выделение и освобождение памяти может занимать много времени и ресурсов, особенно если массив большой. Чтобы избежать этой проблемы, можно использовать другие структуры данных, которые позволяют эффективно добавлять и удалять элементы, например, связный список.
3. Нужна проверка наличия элемента. Прежде чем удалять элемент из массива, необходимо проверить, существует ли он в массиве. Если элемента нет в массиве, программа может завершиться аварийно или удалить неправильный элемент.
4. Необходимость сохранения порядка элементов. Если порядок элементов в массиве важен, необходимо убедиться, что он сохраняется при удалении элементов. В некоторых случаях может быть лучше использовать другую структуру данных, которая позволяет сохранять порядок элементов, например, упорядоченный массив или дерево.
5. Необходимость правильной реализации копирования. При копировании элементов массива необходимо использовать правильную реализацию копирования, чтобы избежать утечек памяти или неожиданного поведения программы.

Пример:

….

struct Initial {

string surname,

name,

patronymic;

};

struct Date {

int day,

month,

year;

};

struct Data {

Initial \_initial;

Date \_date;

};

void Copy(Data\* (&d\_n), Data\* (&d\_o), int n)

{

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

d\_n[i] = d\_o[i];

}

}

void DeleteData(Data\* (&d), int& n)

{

int \_n;

cout << "Введите номер элемента (от 1 до " << n << "): ";

cin >> \_n;

\_n--;

system("cls");

if (\_n >= 0 && \_n < n) {

// временный массив

Data\* buf = new Data[n];

Copy(buf, d, n);

// выделяем новую память

--n;

d = new Data[n];

int q = 0;

// заполняем неудаленные данные

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

if (i != \_n) {

d[q] = buf[i];

++q;

}

}

system("cls");

delete[]buf;

cout << "Данные удалены!" << endl;

}

else

cout << "Номер введён неверно!" << endl;

}

int main()

{

….

// переменная хранящая кол-во структур массива

….

// массив данных

….

//ввод из файла

….

}

В данном и последующих примерах данные вводятся через функцию ReadingData() (из лабораторной работы №5) из файла Input.txt.

Для удаления данных из массива используется функция DeleteData, которая принимает ссылку на массив структур и количество элементов в массиве. Функция запрашивает у пользователя номер элемента, который нужно удалить, после чего создает временный массив, копирует в него все элементы из исходного массива, кроме того, который нужно удалить, выделяет новую память для исходного массива, копирует все элементы из временного массива в исходный массив и освобождает память, занимаемую временным массивом.

Также в программе используется функция Copy, которая копирует данные из одного массива в другой.

1. Функция полного удаления

Для того, чтобы полностью удалить данные нам не нужна новая функция, для этого достаточно в main очистить динамический массив с помощью delete. И создать новый динамический массив.

amountOfData = 0;

delete[]d;

d = new Data[amountOfData];

**Функция изменения:**

….

struct Initial {

string surname,

name,

patronymic;

};

struct Date {

int day,

month,

year;

};

struct Data {

Initial \_initial;

Date \_date;

};

void Copy(Data\* (&d\_n), Data\* (&d\_o), int n)

{

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

d\_n[i] = d\_o[i];

}

}

void DataChange(Data\* (&d), int n)

{

int \_n;

cout << "Введите номер элемента (от 1 до " << n << "): ";

cin >> \_n;

\_n--;

system("cls");

// проверка, что ввели правильное значение

if (\_n >= 0 && \_n < n) {

cout << "Введите ФИО: ";

cin >> d[\_n].\_initial.surname;

cin >> d[\_n].\_initial.name;

cin >> d[\_n].\_initial.patronymic;

cout << "Введите дату: ";

cin >> d[\_n].\_date.day;

cin >> d[\_n].\_date.month;

cin >> d[\_n].\_date.year;

system("cls");

cout << "Данные изменены!" << endl;

}

else

cout << "Номер введён неверно!" << endl;

}

int main()

{

// переменная хранящая кол-во структур массива

….

// массив данных

….

//ввод из файла

….

}

в данном коде есть функция DataChange, которая позволяет изменять данные в массиве структур Data. Эта функция принимает в качестве параметра указатель на массив структур Data и количество элементов в этом массиве. Затем функция запрашивает у пользователя номер элемента, который нужно изменить, и запрашивает новые значения для ФИО и даты. После ввода новых данных функция выводит сообщение об успешном изменении данных.

Решение ОСНОВНЫХ ЗАДАЧ

**Условие:**

Написать функции выборосного и полного удаления массива структур для вашей бд, а также функцию изменения массива структур для вашей БД (из лабораторной работы №2).

**Решение:**

*#include* <iostream>

*#include* <vector>

*#include* <fstream>

*#include* <sstream>

*#include* <algorithm>

*using* *namespace* std;

*struct* *Phone* {

*int* id;

*string* model;

*double* price;

};

*struct* *Teacher* {

*int* id;

*string* name;

vector*<string>* subjects;

vector*<int>* phoneIds;

};

vector*<Teacher>* teachers;

vector*<Phone>* phones;

*void* loadDataFromFile(*const* *string&* teacherFile, *const* *string&* phoneFile, *const* *string&* foreigingkeysFile) {

*ifstream* teacherStream(teacherFile);

*ifstream* phoneStream(phoneFile);

*ifstream* foreigingkeysStream(foreigingkeysFile);

*string* line;

*while* (getline(phoneStream, line)) {

*stringstream* ss(line);

*int* id;

*string* model;

*double* price;

ss >> id >> model >> price;

phones.push\_back({ id, model, price });

}

*while* (getline(teacherStream, line)) {

*stringstream* ss(line);

*int* id;

*string* name;

*int* subjectCount;

ss >> id >> name >> subjectCount;

*Teacher* teacher;

teacher.id *=* id;

teacher.name = name;

*for* (*int* i *=* 0; i *<* subjectCount; *++*i) {

*string* subject;

ss >> subject;

teacher.subjects.push\_back(subject);

}

teachers.push\_back(teacher);

}

*while* (getline(foreigingkeysStream, line)) {

*stringstream* ss(line);

*int* teacherId, phoneId;

ss >> teacherId >> phoneId;

*for* (*auto&* teacher : teachers) {

*if* (teacher.id *==* teacherId) {

teacher.phoneIds.push\_back(phoneId);

*break*;

}

}

}

cout << "Data loaded from files successfully!" << endl;

}

*void* findPhoneByPriceLinear(*double* price) {

cout << "Searching phone with price: " << price << endl;

*for* (*auto&* p : phones) {

*if* (p.price *==* price) {

cout << "Found: Model: " << p.model << " | Price: " << p.price << "$" << endl;

*return*;

}

}

cout << "Phone with the given price not found." << endl;

}

*void* findPhoneByPriceBinary(*double* price) {

vector*<Phone>* sortedPhones *=* phones;

sort(sortedPhones.begin(), sortedPhones.end(), [](*const* *Phone&* a, *const* *Phone&* b) {

*return* a.price *<* b.price;

});

cout << "Searching phone with price: " << price << " (Binary Search)" << endl;

*int* low *=* 0, high *=* sortedPhones.size() *-* 1;

*while* (low *<=* high) {

*int* mid *=* low *+* (high *-* low) */* 2;

*if* (sortedPhones[mid].price *==* price) {

cout << "Found: Model: " << sortedPhones[mid].model << " | Price: " << sortedPhones[mid].price << "$" << endl;

*return*;

}

*if* (sortedPhones[mid].price *<* price) {

low *=* mid *+* 1;

}

*else* {

high *=* mid *-* 1;

}

}

cout << "Phone with the given price not found." << endl;

}

*void* findPhoneByPriceInterpolation(*double* price) {

vector*<Phone>* sortedPhones *=* phones;

sort(sortedPhones.begin(), sortedPhones.end(), [](*const* *Phone&* a, *const* *Phone&* b) {

*return* a.price *<* b.price;

});

cout << "Searching phone with price: " << price << " (Interpolation Search)" << endl;

*int* low *=* 0, high *=* sortedPhones.size() *-* 1;

*while* (low *<=* high *&&* sortedPhones[low].price *<=* price *&&* sortedPhones[high].price *>=* price) {

*int* pos *=* low *+* ((price *-* sortedPhones[low].price) *\** (high *-* low)) */* (sortedPhones[high].price *-* sortedPhones[low].price);

*if* (pos *<* low *||* pos *>* high) {

*break*;

}

*if* (sortedPhones[pos].price *==* price) {

cout << "Found: Model: " << sortedPhones[pos].model << " | Price: " << sortedPhones[pos].price << "$" << endl;

*return*;

}

*if* (sortedPhones[pos].price *<* price) {

low *=* pos *+* 1;

}

*else* {

high *=* pos *-* 1;

}

}

cout << "Phone with the given price not found." << endl;

}

*void* printInfo() {

*for* (*const* *auto&* teacher : teachers) {

cout << "Teacher ID: " << teacher.id << " | Name: " << teacher.name << " | Subjects: ";

*for* (*const* *auto&* subject : teacher.subjects) {

cout << subject << " ";

}

cout << "| Phones: ";

*for* (*int* phoneId : teacher.phoneIds) {

*for* (*const* *auto&* phone : phones) {

*if* (phone.id *==* phoneId) {

cout << phone.model << " (Price: " << phone.price << "$) ";

*break*;

}

}

}

cout << endl;

}

}

*void* printAllPhones() {

*for* (*const* *auto&* phone : phones) {

cout << "Phone ID: " << phone.id << " | Model: " << phone.model << " | Price: " << phone.price << "$" << endl;

}

}

*void* findTeacherByName(*const* *string&* name) {

*bool* found *=* *false*;

*for* (*const* *auto&* teacher : teachers) {

*if* (teacher.name == name) {

cout << "Found Teacher: ID: " << teacher.id << " | Name: " << teacher.name << " | Subjects: ";

*for* (*const* *auto&* subject : teacher.subjects) {

cout << subject << " ";

}

cout << endl;

found *=* *true*;

*break*;

}

}

*if* (*!*found) {

cout << "Teacher with the given name not found." << endl;

}

}

*int* main() {

*int* choice;

loadDataFromFile("teachers.txt", "phones.txt", "foreigingkeys.txt");

*while* (*true*) {

cout << "\nUniversity System\n"

<< "1. Search Phone by Price (Linear Search)\n"

<< "2. Search Phone by Price (Binary Search)\n"

<< "3. Search Phone by Price (Interpolation Search)\n"

<< "4. Print printInfo\n"

<< "5. Search Teacher by Name\n"

<< "9. Exit\n"

<< "Choose option: ";

cin >> choice;

cin.ignore();

*double* price;

*string* name;

*switch* (choice) {

*case* 1: {

cout << "Enter price: ";

cin >> price;

findPhoneByPriceLinear(price);

*break*;

}

*case* 2: {

cout << "Enter price: ";

cin >> price;

findPhoneByPriceBinary(price);

*break*;

}

*case* 3: {

cout << "Enter price: ";

cin >> price;

findPhoneByPriceInterpolation(price);

*break*;

}

*case* 4:

printInfo();

*break*;

*case* 5:

cout << "Enter teacher name: ";

getline(cin, name);

findTeacherByName(name);

*break*;

*case* 9:

*return* 0;

*default*:

cout << "Invalid choice!" << endl;

}

}

}