Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №2

на тему

**РАБОТА С ФАЙЛАМИ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc182478681)

[1.1 Постановка задачи 3](#_Toc182478682)

[1.2 Демонстрация работы программы 3](#_Toc182478683)

[Заключение 6](#_Toc182478684)

[Приложение А (справочное) Исходный код  
 (к пункту 1.2) 8](#_Toc182478685)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

* 1. **Постановка задачи**

В данной лабораторной работе необходимо разработать упрощенную систему управления базами данных с использованием отображения файла в память. Основной задачей является создание функционала для добавления, удаления и отображения структурированных данных, а также управление метаданными и обеспечением эффективного доступа к записям.

Целью работы является создание базы данных, способной хранить до 200 записей с возможностью логического удаления объектов. Важно реализовать возможность добавления, удаления и отображения записей.

Для достижения этих целей нужно будет сначала определить структуру записи с полями для *ID*, имени и флага удаления. Затем необходимо реализовать функцию *CreateDatabase*, которая создаст файл базы данных и инициализирует его пустыми записями. Важно обеспечить отображение файла в память [1] с помощью функций *CreateFileW* [2] и *MapViewOfFile* [3], что обеспечит эффективный доступ к данным.

Далее потребуется реализовать функции для добавления новых записей с уникальным *ID* и функцию удаления записей с использованием логического удаления, позволяя помечать записи как удаленные. Необходимо также создать механизм для отображения всех существующих записей, игнорируя при этом удаленные. Реализация возможности увеличения размера файла базы данных также является важной частью задачи.

Программа будет представлять собой консольное приложение, которое предоставляет пользователю возможность управлять записями в базе данных. Способы доступа к данным будут реализованы через отображение файла в память, что обеспечит высокую скорость работы с данными. В отличие от синхронных операций ввода-вывода, данная реализация позволит избежать блокировок, связанных с традиционными подходами, обеспечивая более высокую производительность и отзывчивость приложения. Упрощенная база данных, разработанная в данной лабораторной работе, будет эффективным инструментом для хранения структурированных данных с возможностью динамического управления записями и оптимизации использования пространства. Основное внимание будет уделено простоте использования и эффективности операций.

**1.2 Демонстрация работы программы**

Консольное приложение при запуске предлагает выбрать один из пунктов меню (рисунок 1.1).



Рисунок 1.1 – Меню консольного приложения

При добавлении записи предлагается ввести *ID* и имя новой записи (рисунок 1.2).

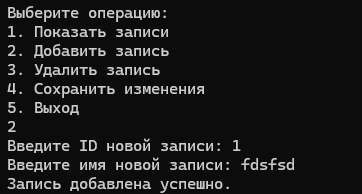


Рисунок 1.2 – Добавление новой записи

Можно просмотреть записи, выбрав соответствующий пункт меню (рисунок 1.3).

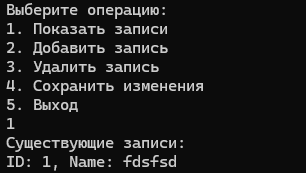


Рисунок 1.3 – Отображение записей

Выбрав пункт удаления записи, программа запросит *ID* и удалит соответствующую запись (рисунок 1.4).

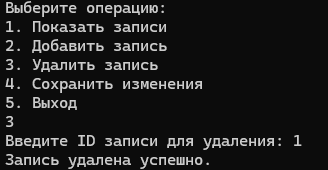


Рисунок 1.4 – Удаление записи

Также записи можно сохранить в файл, выбрав четвертый пункт меню (рисунок 1.5).

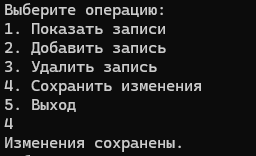


Рисунок 1.5 – Сохранение записей в файл

В данном разделе было представлено консольное приложение для управления записями в базе данных. Программа демонстрирует возможность работы с файлами через отображение в память, что обеспечивает эффективное взаимодействие с данными.

**ЗАКЛЮЧЕНИЕ**

В заключение, разработанная упрощенная система управления базами данных демонстрирует эффективный подход к хранению и обработке структурированных данных с использованием механизма отображения файлов в память. Реализованные функции для добавления, удаления и отображения записей позволяют пользователю удобно управлять данными, а логическое удаление обеспечивает эффективное использование пространства без необходимости физического удаления записей.

Интерфейс программы обеспечивает простоту и удобство взаимодействия, позволяя пользователю легко выполнять необходимые операции. Обработка ошибок и валидация ввода добавляют надежности в работу системы.

Таким образом, выполненная работа не только достигла поставленных целей, но и создала основу для дальнейшего развития и улучшения функциональности базы данных, что может быть полезно в различных приложениях, требующих эффективного управления данными.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Файлы, отображаемые в память [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/55716/.

[2] CreateFileW function (fileapi.h) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilew.

[3] MapViewOfFile function (memoryapi.h) [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/memoryapi  
/nf-memoryapi-mapviewoffile.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

#include <iostream>

#include <windows.h>

#include <vector>

#include <string>

const std::wstring FILENAME = L"database.dat";

const int MAX\_RECORDS = 200;

struct Record {

int id;

wchar\_t name[50];

bool isDeleted;

};

HANDLE hFile;

HANDLE hMapFile;

Record\* pMapView;

void CreateDatabase() {

hFile = CreateFileW(FILENAME.c\_str(), GENERIC\_READ | GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

std::cerr << "Error creating file: " << GetLastError() << std::endl;

return;

}

DWORD bytesWritten;

std::vector<Record> records(MAX\_RECORDS);

for (int i = 0; i < MAX\_RECORDS; ++i) {

records[i].isDeleted = true;

}

WriteFile(hFile, records.data(), sizeof(Record) \* MAX\_RECORDS, &bytesWritten, NULL);

CloseHandle(hFile);

}

void MapDatabase() {

hFile = CreateFileW(FILENAME.c\_str(), GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile == INVALID\_HANDLE\_VALUE) {

std::cerr << "Error opening file: " << GetLastError() << std::endl;

return;

}

hMapFile = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, sizeof(Record) \* MAX\_RECORDS, NULL);

if (hMapFile == NULL) {

std::cerr << "Could not create file mapping object: " << GetLastError() << std::endl;

CloseHandle(hFile);

return;

}

pMapView = (Record\*)MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, sizeof(Record) \* MAX\_RECORDS);

if (pMapView == NULL) {

std::cerr << "Could not map view of file: " << GetLastError() << std::endl;

CloseHandle(hMapFile);

CloseHandle(hFile);

return;

}

}

void DisplayRecords() {

std::wcout << L"Существующие записи:\n";

for (int i = 0; i < MAX\_RECORDS; ++i) {

if (!pMapView[i].isDeleted) {

std::wcout << L"ID: " << pMapView[i].id << L", Name: " << pMapView[i].name << std::endl;

}

}

}

void AddRecord(int id, const wchar\_t\* name) {

for (int i = 0; i < MAX\_RECORDS; ++i) {

if (pMapView[i].isDeleted) {

pMapView[i].id = id;

wcscpy\_s(pMapView[i].name, name);

pMapView[i].isDeleted = false;

std::cout << "Запись добавлена успешно." << std::endl;

return;

}

}

std::cout << "Нет места для добавления новой записи." << std::endl;

}

void DeleteRecord(int id) {

for (int i = 0; i < MAX\_RECORDS; ++i) {

if (pMapView[i].id == id && !pMapView[i].isDeleted) {

pMapView[i].isDeleted = true;

std::cout << "Запись удалена успешно." << std::endl;

return;

}

}

std::cout << "Запись не найдена." << std::endl;

}

void SaveChanges() {

DWORD bytesWritten;

SetFilePointer(hFile, 0, NULL, FILE\_BEGIN);

WriteFile(hFile, pMapView, sizeof(Record) \* MAX\_RECORDS, &bytesWritten, NULL);

}

void Cleanup() {

UnmapViewOfFile(pMapView);

CloseHandle(hMapFile);

CloseHandle(hFile);

}

void ShowMenu() {

std::cout << "Выберите операцию:\n";

std::cout << "1. Показать записи\n";

std::cout << "2. Добавить запись\n";

std::cout << "3. Удалить запись\n";

std::cout << "4. Сохранить изменения\n";

std::cout << "5. Выход\n";

}

int main() {

setlocale(LC\_ALL, "Russian");

CreateDatabase();

MapDatabase();

int choice;

while (true) {

ShowMenu();

std::cin >> choice;

switch (choice) {

case 1:

DisplayRecords();

break;

case 2: {

int id;

wchar\_t name[50];

std::cout << "Введите ID новой записи: ";

std::cin >> id;

std::wcout << L"Введите имя новой записи: ";

std::wcin.ignore();

std::wcin.getline(name, 50);

AddRecord(id, name);

break;

}

case 3: {

int id;

std::cout << "Введите ID записи для удаления: ";

std::cin >> id;

DeleteRecord(id);

break;

}

case 4:

SaveChanges();

std::cout << "Изменения сохранены." << std::endl;

break;

case 5:

Cleanup();

return 0;

default:

std::cout << "Неверный выбор. Пожалуйста, попробуйте снова." << std::endl;

}

}

}