ЛАБОРАТОРНАЯ РАБОТА 2.
ОПЕРАЦИИ С ФАЙЛАМИ В WINDOWS

Цель работы: изучение операций с файлами в операционной системе Windows с использованием системных вызовов и API. Приобретение навыков работы с файлами на низком уровне, включая создание, чтение, запись, удаление и управление файловыми атрибутами.

Основные теоретические сведения

В Windows API (WinAPI) существует множество функций для работы с файлами и каталогами. Рассмотрим примеры работы с наиболее распространенными из них.

Пример кода на C/C++ для открытия файла, чтения из него и закрытия файла с использованием WinAPI:

```
#include <windows.h>
#include "iostream"

int main() {
    SetConsoleOutputCP(1251);
    // Открыть файл для чтения
    HANDLE hFile = CreateFile("example.txt", GENERIC_READ, 0, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        // Чтение данных из файла
        char buffer[1024];
        DWORD bytesRead;
        if (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL)) {
            // Обработка данных
        }

        // Закрыть файл
        CloseHandle(hFile);
    } else {
        // Обработка ошибки открытия файла
        std::cout << "Не удалось открыть файл" << std::endl;
    }

    return 0;
}
```

Создание файла в Windows с использованием WinAPI можно выполнить с помощью функции `CreateFile`. Она имеет множество параметров, которые позволяют настроить операцию создания или открытия файла подробно. Общий вид прототипа функции:

```
HANDLE CreateFile(
  LPCTSTR               lpFileName,
  DWORD                 dwDesiredAccess,
  DWORD                 dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD                 dwCreationDisposition,
  DWORD                 dwFlagsAndAttributes,
  HANDLE                hTemplateFile
);
```

- **lpFileName (LPCTSTR)**: представляет собой путь к файлу или устройству. Может быть строкой в формате многобайтовых символов (ANSI) или широких символов (Unicode), в зависимости от того, какой тип используется: LPCSTR для ANSI или LPCWSTR для Unicode.

Примеры:

ANSI: `"C:\\example.txt"`

Unicode: `L"C:\\example.txt"`

- **dwDesiredAccess (DWORD):** определяет режим доступа к файлу. Это флаги, которые указывают, какие операции можно выполнять с файлом. Например:
  - GENERIC_READ: разрешает чтение файла.
  - GENERIC_WRITE: разрешает запись в файл.

Можно комбинировать флаги с помощью операции побитового ИЛИ (|) для указания нескольких прав доступа.

- **dwShareMode (DWORD):** определяет режим совместного доступа к файлу. Это флаги, указывающие, какие типы доступа можно разрешить другим процессам. Например:
  - FILE_SHARE_READ: разрешает другим процессам читать файл.
  - FILE_SHARE_WRITE: разрешает другим процессам записывать в файл.
- **lpSecurityAttributes (LPSECURITY_ATTRIBUTES):** позволяет указать атрибуты безопасности для создаваемого файла. Если вы не хотите задавать специальные атрибуты безопасности, можете передать NULL.
- **dwCreationDisposition (DWORD):** определяет, что делать, если файл уже существует или не существует. Некоторые из возможных значений:
  - CREATE_NEW: создать новый файл (ошибка, если файл уже существует).
  - CREATE_ALWAYS: создать новый файл или перезаписать существующий.
  - OPEN_EXISTING: открыть только существующий файл.
  - OPEN_ALWAYS: открыть существующий файл или создать новый, если его нет.
- **dwFlagsAndAttributes (DWORD):** позволяет указать дополнительные атрибуты файла. Например:
  - FILE_ATTRIBUTE_NORMAL: обычный файл без специальных атрибутов.
  - FILE_ATTRIBUTE_READONLY: файл только для чтения.
- **hTemplateFile (HANDLE):** представляет собой дескриптор файла-шаблона, который используется для определения атрибутов и флагов создаваемого файла. Обычно передается как NULL.

После вызова функции `CreateFile`, она возвращает дескриптор файла (или специальное значение `INVALID_HANDLE_VALUE` в случае ошибки). Этот

дескриптор файла используется для дальнейших операций с файлом, таких как чтение, запись и закрытие.

Пример демонстрирующий, как создать новый файл:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    // Открыть или создать файл для записи
    HANDLE hFile = CreateFile(fileName, GENERIC_WRITE, 0, NULL, CREATE_NEW,
FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        // Файл успешно создан
        std::cout << "Файл '" << fileName << "' успешно создан." << std::endl;

        // Закрыть файл
        CloseHandle(hFile);
    } else {
        // Обработать ошибку
        DWORD error = GetLastError();
        if (error == ERROR_FILE_EXISTS) {
            std::cout << "Файл '" << fileName << "' уже существует." <<
std::endl;
        } else {
            std::cerr << "Не удалось создать файл '" << fileName << "'. Ошибка
" << error << std::endl;
        }
    }

    return 0;
}
```

Функция `ReadFile` в Windows API используется для чтения данных из файла. Её прототип:

```
BOOL ReadFile(
  HANDLE       hFile,
  LPVOID       lpBuffer,
  DWORD        nNumberOfBytesToRead,
  LPDWORD      lpNumberOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
```

- **`hFile (HANDLE)`:** дескриптор файла, который был получен при открытии файла с помощью функции `CreateFile`. Указывает на файл, из которого будут читаться данные.

- **lpBuffer (LPVOID):** указатель на буфер, в который будут считываться данные из файла. Этот буфер должен быть предварительно выделен, и его размер должен быть не меньше, чем `nNumberOfBytesToRead`, чтобы вместить считанные данные.
- **nNumberOfBytesToRead (DWORD):** указывает количество байт, которые нужно прочитать из файла и поместить в буфер `lpBuffer`.
- **lpNumberOfBytesRead (LPDWORD):** указатель на переменную, в которую будет записано фактическое количество байт, которые были прочитаны из файла. Эта информация может быть полезной, чтобы узнать, сколько данных было считано.
- **lpOverlapped (LPOVERLAPPED):** используется для асинхронного чтения файлов и обычно оставляется равным `NULL` для синхронного чтения.

После вызова функции `ReadFile`, она возвращает `TRUE`, если чтение было успешным, и `FALSE`, если произошла ошибка. Если чтение было успешным, данные будут доступны в буфере `lpBuffer`, и количество считанных байт будет записано в переменную, на которую указывает `lpNumberOfBytesRead`.

Пример, демонстрирующий, как читать данные из файла:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    // Открыть файл для чтения
    HANDLE hFile = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        char buffer[1024];
        DWORD bytesRead;

        if (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL)) {
            // Чтение прошло успешно
            if (bytesRead > 0) {
                // Вывести прочитанные данные на экран
                std::cout << "Прочитано " << bytesRead << " байт: " << buffer
<< std::endl;
            } else {
                std::cout << "Файл пуст." << std::endl;
            }
        } else {
            // Обработать ошибку чтения
            DWORD error = GetLastError();
            std::cerr << "Ошибка чтения файла. Код ошибки: " << error <<
std::endl;
```

```
        }

        // Закрыть файл
        CloseHandle(hFile);
    } else {
        // Обработать ошибку открытия файла
        DWORD error = GetLastError();
        std::cerr << "Не удалось открыть файл. Код ошибки: " << error <<
std::endl;
    }

    return 0;
}
```

Запись данных в файл в Windows с использованием WinAPI выполняется с помощью функции `WriteFile`. Эта функция позволяет записать данные из буфера в файл. Её прототип:

```
BOOL WriteFile(
  HANDLE       hFile,
  LPCVOID      lpBuffer,
  DWORD        nNumberOfBytesToWrite,
  LPDWORD      lpNumberOfBytesWritten,
  LPOVERLAPPED lpOverlapped
);
```

- **hFile (HANDLE):** дескриптор файла, указывающий на файл, в который будут записываться данные.
- **lpBuffer (LPCVOID):** указатель на буфер, из которого будут записываться данные в файл. Этот буфер должен содержать данные, которые вы хотите записать.
- **nNumberOfBytesToWrite (DWORD):** параметр указывает количество байт, которые нужно записать в файл из буфера `lpBuffer`.
- **lpNumberOfBytesWritten (LPDWORD):** указатель на переменную, в которую будет записано фактическое количество байт, которые были успешно записаны в файл.
- **lpOverlapped (LPOVERLAPPED):** параметр используется для асинхронной записи файлов и обычно оставляется равным `NULL` для синхронной записи.

После вызова функции `WriteFile`, она возвращает `TRUE`, если запись была успешной, и `FALSE`, если произошла ошибка. Если запись была успешной, количество фактически записанных байт будет записано в переменную, на которую указывает `lpNumberOfBytesWritten`.

Пример записи данных в файл:

```
#include <windows.h>
#include <iostream>
```

```
int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    // Открыть файл для записи
    HANDLE hFile = CreateFile(fileName, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        const char *data = "Пример записи в файл.";
        DWORD bytesWritten;

        if (WriteFile(hFile, data, strlen(data), &bytesWritten, NULL)) {
            // Запись прошла успешно
            std::cout << "Записано " << bytesWritten << " байт." << std::endl;
        } else {
            // Обработать ошибку записи
            DWORD error = GetLastError();
            std::cerr << "Ошибка записи в файл. Код ошибки: " << error <<
std::endl;
        }

        // Закрыть файл
        CloseHandle(hFile);
    } else {
        // Обработать ошибку открытия файла
        DWORD error = GetLastError();
        std::cerr << "Не удалось открыть/создать файл. Код ошибки: " << error
<< std::endl;
    }

    return 0;
}
```

Для удаления файла в Windows с использованием WinAPI можно использовать функцию `DeleteFile`. Её прототип:

```
BOOL DeleteFile(
  LPCTSTR lpFileName
);
```

- **lpFileName (LPCTSTR):** представляет собой строку, содержащую путь к файлу, который вы хотите удалить. Обычно это Unicode-строка, представленная как LPCWSTR для широких символов или LPCTSTR для многобайтовых символов.

Пример использования функции `DeleteFile` для удаления файла:

```
#include <windows.h>
#include <iostream>

int main() {
```

```
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    if (DeleteFile(fileName)) {
        std::cout << "Файл '" << fileName << "' успешно удалён." << std::endl;
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось удалить файл '" << fileName << "'. Код ошибки:
" << error << std::endl;
    }

    return 0;
}
```

Для переименования и перемещения файла в Windows с использованием WinAPI, можно воспользоваться функцией `MoveFile` или её вариациями. Например:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR oldFileName = "C:\\путь\\к\\старому\\файлу\\old_file.txt"; //
Замените старый путь и имя файла
    LPCSTR newFileName = "C:\\путь\\к\\новому\\файлу\\new_file.txt"; //
Замените новый путь и имя файла

    if (MoveFile(oldFileName, newFileName)) {
        std::cout << "Файл успешно переименован в '" << newFileName << "'." <<
std::endl;
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось переименовать файл. Код ошибки: " << error <<
std::endl;
    }

    return 0;
}
```

Можно управлять атрибутами файла в Windows с использованием WinAPI с помощью функций `GetFileAttributes` и `SetFileAttributes`.

Пример получения атрибутов файла:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению
```

```
    DWORD fileAttributes = GetFileAttributes(fileName);

    if (fileAttributes != INVALID_FILE_ATTRIBUTES) {
        if (fileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            std::cout << "Файл '" << fileName << "' - это каталог." <<
std::endl;
        } else {
            std::cout << "Файл '" << fileName << "' - это обычный файл." <<
std::endl;
        }

        if (fileAttributes & FILE_ATTRIBUTE_READONLY) {
            std::cout << "Файл '" << fileName << "' доступен только для
чтения." << std::endl;
        }
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось получить атрибуты файла. Код ошибки: " <<
error << std::endl;
    }

    return 0;
}
```

Пример установки атрибутов файла:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    // Получить текущие атрибуты файла
    DWORD fileAttributes = GetFileAttributes(fileName);

    if (fileAttributes != INVALID_FILE_ATTRIBUTES) {
        // Установить новые атрибуты (например, сделать файл доступным только
для чтения)
        fileAttributes |= FILE_ATTRIBUTE_READONLY;

        if (SetFileAttributes(fileName, fileAttributes)) {
            std::cout << "Атрибуты файла '" << fileName << "' успешно
изменены." << std::endl;
        } else {
            DWORD error = GetLastError();
            std::cerr << "Не удалось изменить атрибуты файла. Код ошибки: " <<
error << std::endl;
        }
    } else {
        DWORD error = GetLastError();
```

```
        std::cerr << "Не удалось получить атрибуты файла. Код ошибки: " <<
error << std::endl;
    }

    return 0;
}
```

Для проверки существования файла в Windows с использованием WinAPI можно воспользоваться функцией `PathFileExists` из библиотеки `Shlwapi.h` или функцией `GetFileAttributes` и проверкой возвращаемого значения.

Пример использования `PathFileExists`:

Для компиляции примера в Code::Blocks необходимо подключить библиотеку в настройках компилятора:

1. Project → Build options → Linker settings → Add
2. Добавить «shlwapi».

```
#include <windows.h>
#include <Shlwapi.h> // Для PathFileExists

int main() {
    SetConsoleOutputCP(1251);
    PCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    if (PathFileExists(fileName)) {
        // Файл существует
        printf("Файл '%s' существует.\n", fileName);
    } else {
        // Файл не существует
        printf("Файл '%s' не существует.\n", fileName);
    }

    return 0;
}
```

Пример использования `GetFileAttributes`:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    DWORD fileAttributes = GetFileAttributes(fileName);

    if (fileAttributes != INVALID_FILE_ATTRIBUTES && !(fileAttributes &
FILE_ATTRIBUTE_DIRECTORY)) {
        // Файл существует
        std::cout << "Файл '" << fileName << "' существует." << std::endl;
    } else {
```

```
        // Файл не существует или это каталог
        std::cout << "Файл '" << fileName << "' не существует или это каталог."
<< std::endl;
    }

    return 0;
}
```

Для поиска файлов в Windows с использованием WinAPI, можно воспользоваться функцией `FindFirstFile` и её итеративной версией `FindNextFile`. Эти функции позволяют искать файлы по определенным критериям в указанном каталоге. Пример поиска файлов:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR searchPath = "C:\\путь\\к\\каталогу\\*.*"; // Замените путь к
каталогу по вашему усмотрению

    WIN32_FIND_DATA findFileData;
    HANDLE hFind = FindFirstFile(searchPath, &findFileData);

    if (hFind != INVALID_HANDLE_VALUE) {
        do {
            if (!(findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {
                // Элемент найден, и это не каталог
                std::cout << "Имя файла: " << findFileData.cFileName <<
std::endl;
            }
        } while (FindNextFile(hFind, &findFileData) != 0);

        FindClose(hFind); // Закрыть дескриптор поиска
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось выполнить поиск файлов. Код ошибки: " << error
<< std::endl;
    }

    return 0;
}
```

Для работы с каталогами в Windows с использованием WinAPI вы можете использовать функции, такие как `CreateDirectory`, `RemoveDirectory`, `SetCurrentDirectory`, и `GetCurrentDirectory`.

Пример создания каталога:

```
#include <windows.h>
#include <iostream>

int main() {
```

```
    SetConsoleOutputCP(1251);
    LPCSTR directoryName = "C:\\путь\\к\\новому\\каталогу"; // Замените путь и
имя каталога по вашему усмотрению

    if (CreateDirectory(directoryName, NULL)) {
        std::cout << "Каталог '" << directoryName << "' успешно создан." <<
std::endl;
    } else {
        DWORD error = GetLastError();
        if (error == ERROR_ALREADY_EXISTS) {
            std::cerr << "Каталог '" << directoryName << "' уже существует." <<
std::endl;
        } else {
            std::cerr << "Не удалось создать каталог. Код ошибки: " << error <<
std::endl;
        }
    }

    return 0;
}
```

Пример удаления каталога:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR directoryName = "C:\\путь\\к\\каталогу\\для_удаления"; // Замените
путь и имя каталога по вашему усмотрению

    if (RemoveDirectory(directoryName)) {
        std::cout << "Каталог '" << directoryName << "' успешно удалён." <<
std::endl;
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось удалить каталог. Код ошибки: " << error <<
std::endl;
    }

    return 0;
}
```

Установка текущего каталога:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR newDirectory = "C:\\путь\\к\\новому\\каталогу"; // Замените путь и имя
каталога по вашему усмотрению

    if (SetCurrentDirectory(newDirectory)) {
        std::cout << "Текущий каталог установлен в '" << newDirectory << "'." <<
std::endl;
```

```
        } else {
            DWORD error = GetLastError();
            std::cerr << "Не удалось установить текущий каталог. Код ошибки: " <<
    error << std::endl;
        }

        return 0;
    }
```

Пример получения текущего каталога:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    CHAR currentDirectory[MAX_PATH];

    if (GetCurrentDirectory(MAX_PATH, currentDirectory) != 0) {
        std::cout << "Текущий каталог: " << currentDirectory << std::endl;
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось получить текущий каталог. Код ошибки: " <<
error << std::endl;
    }

    return 0;
}
```

Функции CreateDirectory и RemoveDirectory могут возвращать ошибку ERROR_ALREADY_EXISTS, если каталог уже существует.

Функция `GetFileInformationByHandle` предоставляет информацию о файле на основе его дескриптора. Эта функция предоставляет подробные сведения о файле, такие как размер, атрибуты, время создания и др. Пример использования функции `GetFileInformationByHandle`:

```
#include <windows.h>
#include <iostream>
#include <ctime>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению

    // Открыть файл для получения дескриптора
    HANDLE hFile = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        BY_HANDLE_FILE_INFORMATION fileInfo;
        if (GetFileInformationByHandle(hFile, &fileInfo)) {
            std::cout << "Имя файла: " << fileName << std::endl;
```

```
            std::cout << "Размер файла: " << fileInfo.nFileSizeLow << " байт"
<< std::endl;
            std::cout << "Атрибуты файла: " << fileInfo.dwFileAttributes <<
std::endl;

            SYSTEMTIME sysTime;
            FileTimeToSystemTime(&fileInfo.ftCreationTime, &sysTime);
            std::cout << "Дата создания: " << sysTime.wDay << "." <<
sysTime.wMonth << "." << sysTime.wYear
                << " " << sysTime.wHour << ":" << sysTime.wMinute << ":" <<
sysTime.wSecond << std::endl;

            FileTimeToSystemTime(&fileInfo.ftLastAccessTime, &sysTime);
            std::cout << "Последний доступ: " << sysTime.wDay << "." <<
sysTime.wMonth << "." << sysTime.wYear
                << " " << sysTime.wHour << ":" << sysTime.wMinute << ":" <<
sysTime.wSecond << std::endl;

            FileTimeToSystemTime(&fileInfo.ftLastWriteTime, &sysTime);
            std::cout << "Последнее изменение: " << sysTime.wDay << "." <<
sysTime.wMonth << "."
                << sysTime.wYear << " " << sysTime.wHour << ":" <<
sysTime.wMinute << ":" << sysTime.wSecond << std::endl;
        } else {
            DWORD error = GetLastError();
            std::cerr << "Не удалось получить информацию о файле. Код ошибки: "
<< error << std::endl;
        }

        // Закрыть дескриптор файла
        CloseHandle(hFile);
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось открыть файл. Код ошибки: " << error <<
std::endl;
    }

    return 0;
}
```

Для закрытия файла в Windows с использованием WinAPI, можно использовать функцию `CloseHandle`, передав в неё дескриптор файла, который был получен при его открытии с помощью функции `CreateFile` или других функций, возвращающих дескриптор файла. Пример закрытия файла:

```
#include <windows.h>
#include <iostream>

int main() {
    SetConsoleOutputCP(1251);
    LPCSTR fileName = "C:\\путь\\к\\вашему\\файлу\\example.txt"; // Замените
путь и имя файла по вашему усмотрению
```

```
    // Открыть файл для чтения
    HANDLE hFile = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);

    if (hFile != INVALID_HANDLE_VALUE) {
        // Выполняйте операции с файлом

        // Закрыть дескриптор файла
        CloseHandle(hFile);
        std::cout << "Файл закрыт." << std::endl;
    } else {
        DWORD error = GetLastError();
        std::cerr << "Не удалось открыть файл. Код ошибки: " << error <<
std::endl;
    }

    return 0;
}
```

Порядок выполнения работы

Задание 1. Написать программу на языке C или C++ которая:

- Создает каталог с названием группы.
- Создает в этом каталоге файл с именем, соответствующем вашей фамилии.
- Записывает в этот файл текущую дату.
- Считывает содержимое файла и выводит на экран.
- Переименовывает файл, добавляя к названию инициалы.
- Определяет размер файла и выводит на экран.
- Удаляет ранее созданные файл и каталог.
- Выводит на экран список файлов и каталогов на диске C.

Задание 2. Написать программу на языке C или C++ согласно варианту. Все действия с файлами и каталогами осуществлять с помощью функций WinAPI.

Варианты задания:

1. Создать программу для чтения содержимого текстового файла и подсчета количества слов в нем с использованием WinAPI.
2. Написать утилиту, которая будет сравнивать два текстовых файла и определять, совпадают ли они.
3. Написать программу для поиска и замены заданной строки в текстовом файле с использованием функций WinAPI.
4. Создать программу, которая будет выводить список всех файлов и поддиректорий в заданной директории с указанием их размеров.
5. Написать утилиту для создания списка файлов в заданной директории и сохранения его в текстовом файле.

6. Разработать приложение для сортировки содержимого текстового файла в алфавитном порядке и сохранения результата в новом файле с использованием WinAPI.

7. Создать утилиту для поиска дубликатов файлов в заданной директории и ее поддиректориях.

8. Разработать программу для сравнения содержимого двух текстовых файлов и вывода различий на экран.

9. Написать приложение для поиска всех файлов в заданной директории и ее поддиректориях. Результат поиска должен выводиться на экран.

10. Напишите утилиту для создания списка файлов в заданной директории и сохранения его в текстовом файле.

Контрольные вопросы

1. *Теоретические основы:*

- Что такое файловая система и какие виды файловых систем поддерживаются в Windows?
- Какие основные операции с файлами поддерживаются в Windows API?
- В чем различие между системными вызовами и API-функциями?

2. *Основные API-функции:*

- Какие функции Windows API используются для создания и открытия файла? Приведите примеры.
- Как с помощью Windows API можно прочитать данные из файла и записать данные в файл? Опишите процесс.
- Какие функции используются для удаления файла в Windows?

3. *Работа с атрибутами файлов:*

- Какие атрибуты файлов поддерживаются в Windows?
- Как изменить атрибуты файла с помощью Windows API?
- Какой системный вызов позволяет проверить существование файла и его атрибуты?

4. *Обработка ошибок:*

- Какие наиболее распространенные ошибки могут возникать при работе с файлами в Windows и как их обрабатывать?
- Как в программе на C/C++ отловить и корректно обработать ошибку «файл не найден»?
- Что такое дескриптор файла и как правильно с ним работать, чтобы избежать утечек ресурсов?

5. *Практическая реализация:*

- Какой порядок выполнения операций с файлами в типичной программе на C/C++ с использованием Windows API?
- Приведите пример кода на C/C++, который создает файл, записывает в него данные и затем закрывает его.
- Какие меры предосторожности нужно соблюдать при работе с файлами, чтобы избежать повреждения данных?

6. *Анализ и отладка:*

- Как можно отладить программу, работающую с файлами, для выявления и устранения ошибок?
- Какие инструменты Windows помогают мониторить операции с файлами (например, Process Monitor)?
- Какие существуют методы проверки целостности файлов после выполнения операций записи?