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 №1

на тему

Расширенное использование оконного интерфейса Win 32 и GDI

Выполнил: студент группы 153503

Скворцов Александр Владимирович

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[2 Результаты выполнения лабораторной работы 6](#_Toc146728179)

[Выводы 8](#_Toc146728180)

[Список использованных источников 9](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 10](#_Toc146728182)

## 1 ЦЕЛЬ РАБОТЫ

## Целью выполнения лабораторной работы является расширенное использование оконного интерфейса Win32 и GDI, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, использование механизма перехвата сообщений (winhook). Чтобы продемонстрировать все условия, будет создано оконное приложение «Airplane», позволяющее управлять моделью самолета, изображенного с помощью функций GDI.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) - это набор функций и интерфейсов прикладного программирования, предоставляемых операционной системой Microsoft Windows для разработки приложений под эту операционную систему. Win32 API предоставляет доступ к широкому спектру функциональности операционной системы, включая создание окон, работу с файлами и директориями, многозадачность, ввод и вывод данных, сетевое взаимодействие и многое другое.

Разработчики используют Win32 API на языках программирования C и C++ для создания приложений под операционной системой Windows.

GDI (Graphics Device Interface) - это компонент операционной системы Windows, который предоставляет API для рисования 2D графики и взаимодействия с графическими устройствами, такими как мониторы, принтеры и другие выводящие устройства. GDI обеспечивает базовую функциональность для работы с графическими ресурсами, такими как кисти, перья, шрифты и регионы, и предоставляет множество функций для создания и манипулирования изображениями.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

– WndProc — это процедура обработки сообщений для окна приложения. Она обрабатывает сообщения, поступающие от операционной системы.

– CreateWindow: функция, используемая для создания окна или элемента управления в окне. Принимает параметры, такие как класс окна, заголовок, стиль окна и позицию.

– Обработка сообщений: В главном цикле сообщений (WinMain) приложение ожидает и обрабатывает сообщения с помощью GetMessage, TranslateMessage и DispatchMessage. WndProc обрабатывает различные типы сообщений, такие как рисование, клавиатурные события и таймеры.

– Рисование: Рисование выполняется в процедуре WM\_PAINT. Окно представляет из себя поле для ввода текста.

– Хук клавиатуры: Приложение также демонстрирует использование глобального хука клавиатуры с помощью функций SetWindowsHookEx и UnhookWindowsHookEx.

– Обработка ошибок: В коде предусмотрена обработка ошибок с использованием сообщений MessageBox, чтобы уведомлять пользователя о проблемах.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был разработан текстовый редактор, напоминающий классическое приложение «Блокнот». Приложение позволяет пользователю создать текстовый файл, печатать текст с клавиатуры, поменять шрифт и цвет текста, а также поменять цвет фона текстового поля (рисунок 1).

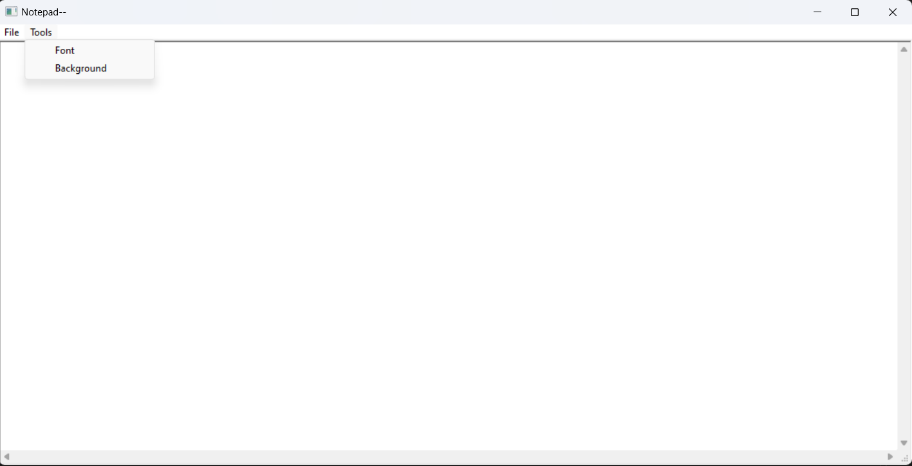


Рисунок 1 - Главный экран

Пользователь может выбрать стили текста, такие как шрифт, начертание, размер, видоизменение и цвет (рисунок 2).

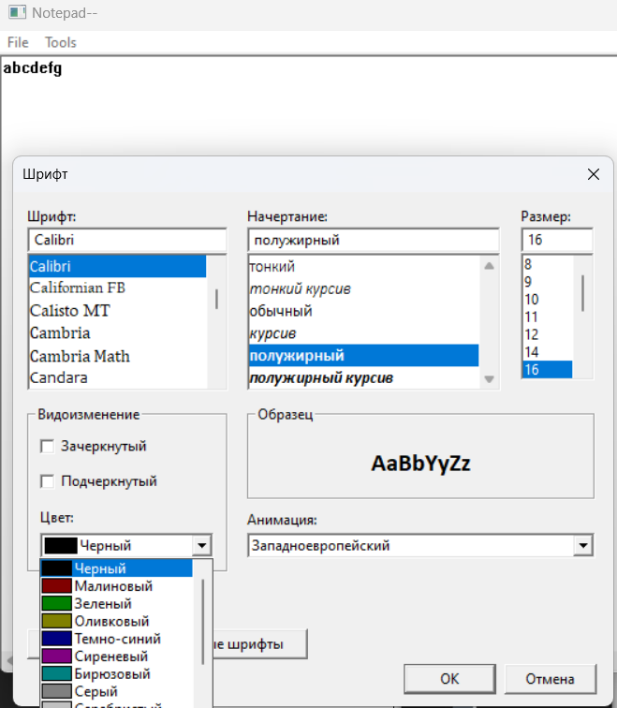


Рисунок 2 ─ Изменение стиля текста

Так же пользователь имеет возможность поменять цвет фона текстового поля (рисунок 3).

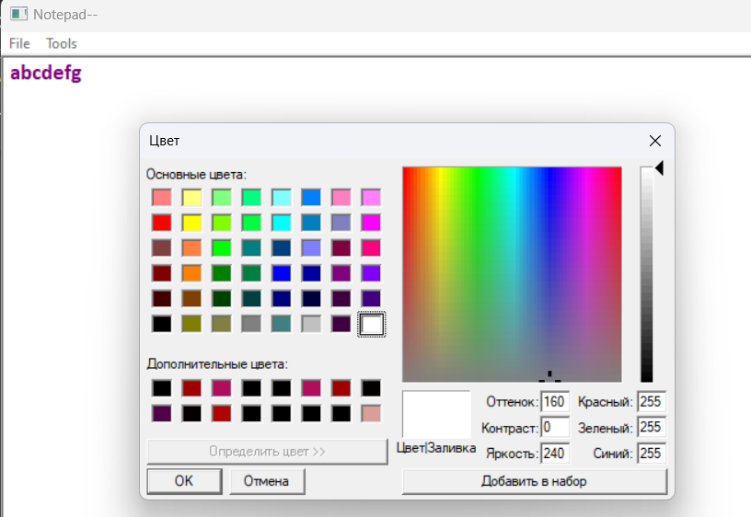


Рисунок 3 – Изменение цвета фона



Рисунок 4 – Измененный стиль редактора

## ВЫВОДЫ

Таким образом, в ходе данной лабораторной работы было проведено расширенное использование оконного интерфейса Win32 и GDI для формирования сложных изображений, создания и использования элементов управления, а также обработки различных сообщений. Был реализован механизм перехвата сообщений с помощью WinHook. В результате работы был разработан текстовый редактор, обладающий возможностью изменения стиля текста и цвета фона. В ходе работы успешно освоилось создание оконного приложения, работу с элементами управления, обработку сообщений и использование WinHook для перехвата сообщений. Изучились принципы работы с графическим интерфейсом и возможности GDI, что позволило нам создавать и редактировать сложные изображения.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ил.

[2] Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 19.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

Листинг 1 – Файл main.cpp

#ifndef UNICODE

#define UNICODE

#endif

#define ID\_FILE\_CREATE 9001

#define ID\_FILE\_OPEN 9002

#define ID\_FILE\_SAVE 9003

#define VK\_A 0x41

#define ID\_FONT\_CHOICE 9004

#define ID\_BG\_CHOICE 9005

// Макросы для распознавания конкретной команды из MenuBar'а

#include <windows.h>

#include <shobjidl.h>

#include <iostream>

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK HotkeyProc(int nCode, WPARAM wParam, LPARAM lParam);

void OpenFile(HWND hwnd);

void SaveFile(HWND hwnd);

void FontChoice(HWND hwnd);

void BackgroundColor(HWND hwnd);

HWND hWndEdit = NULL;

COLORREF hEditFontColor = RGB(0, 0, 0);

HFONT hEditFont = NULL;

COLORREF hEditBackgroundColor = RGB(255, 255, 255);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow) {

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

HHOOK hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, HotkeyProc, NULL, 0);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"Notepad--",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

return 0;

}

ShowWindow(hwnd, nCmdShow);

MSG msg = { };

while (GetMessage(&msg, NULL, 0, 0) > 0) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hHook);

return 0;

}

UINT\_PTR CALLBACK Lpcchookproc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

if (msg == WM\_INITDIALOG) {

SetWindowPos(hWnd, HWND\_TOPMOST, 400, 400, 0, 0, SWP\_NOSIZE);

}

return 0;

}

LRESULT CALLBACK HotkeyProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode < 0)

return CallNextHookEx(NULL, nCode, wParam, lParam);

if (wParam == WM\_KEYDOWN) {

if (nCode >= 0) {

KBDLLHOOKSTRUCT\* pKeyboardHook = (KBDLLHOOKSTRUCT\*)lParam;

if (pKeyboardHook->vkCode == 'S' && GetAsyncKeyState(VK\_CONTROL) & 0x8000 && GetAsyncKeyState(VK\_SHIFT) & 0x8000) {

MessageBox(NULL, L"Вы нажали Ctrl + Shift + S!", L"Горячая клавиша", MB\_OK);

}

}

}

return CallNextHookEx(NULL, nCode, wParam, lParam);

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

case WM\_CREATE:

{

HRESULT hr = CoInitializeEx(NULL, COINIT\_APARTMENTTHREADED | COINIT\_DISABLE\_OLE1DDE);

if (!SUCCEEDED(hr)) {

DestroyWindow(hwnd);

return 0;

}

HMENU hMenu, hFileSubMenu, hToolsSubMenu;

// Создание MenuBar

hMenu = CreateMenu();

hFileSubMenu = CreatePopupMenu();

hToolsSubMenu = CreatePopupMenu();

AppendMenu(hFileSubMenu, MF\_STRING, ID\_FILE\_CREATE, L"&Create");

AppendMenu(hFileSubMenu, MF\_STRING, ID\_FILE\_OPEN, L"&Open");

AppendMenu(hFileSubMenu, MF\_STRING, ID\_FILE\_SAVE, L"&Save");

AppendMenu(hMenu, MF\_POPUP, (UINT)hFileSubMenu, L"&File");

AppendMenu(hToolsSubMenu, MF\_STRING, ID\_FONT\_CHOICE, L"&Font");

AppendMenu(hToolsSubMenu, MF\_STRING, ID\_BG\_CHOICE, L"&Background");

AppendMenu(hMenu, MF\_POPUP, (UINT)hToolsSubMenu, L"&Tools");

SetMenu(hwnd, hMenu);

hWndEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("Edit"), NULL,

WS\_CHILD | WS\_VSCROLL | WS\_HSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL | ES\_LEFT | ES\_MULTILINE | ES\_WANTRETURN | WS\_VISIBLE,

0, 0, 780, 560, hwnd, NULL,

(HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE),

NULL);

hEditFont = (HFONT)SendMessage(hWndEdit, WM\_GETFONT, 0, 0);

RegisterHotKey(hwnd, 1, MOD\_CONTROL, VK\_A);

return 0;

}

case WM\_COMMAND:

{

switch (LOWORD(wParam)) {

// Обработка сообщений из нажатых кнопок в меню

case ID\_FILE\_CREATE:

SetWindowTextA(hWndEdit, "");

break;

case ID\_FILE\_OPEN:

OpenFile(hwnd);

break;

case ID\_FILE\_SAVE:

SaveFile(hwnd);

break;

case ID\_FONT\_CHOICE:

FontChoice(hwnd);

break;

case ID\_BG\_CHOICE:

BackgroundColor(hwnd);

break;

default:

break;

}

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

EndPaint(hwnd, &ps);

return 0;

}

case WM\_SIZE:

{

// Растягивание окна текстового редактора совместно с растягиванием основного окна

MoveWindow(hWndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), false);

return 0;

}

case WM\_CTLCOLOREDIT:

{

HDC hdc = (HDC)wParam;

SetTextColor(hdc, hEditFontColor);

SetBkColor(hdc, hEditBackgroundColor);

SetDCBrushColor(hdc, hEditBackgroundColor);

return (LRESULT)GetStockObject(DC\_BRUSH);

}

case WM\_HOTKEY:

{

if (wParam == 1) {

SendMessage(hWndEdit, EM\_SETSEL, 0, GetWindowTextLength(hWndEdit));

}

break;

}

case WM\_CLOSE:

{

CoUninitialize();

DestroyWindow(hwnd);

return 0;

}

return 0;

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

void OpenFile(HWND hwnd) {

IFileOpenDialog\* pFileOpen;

HRESULT hr = CoCreateInstance(CLSID\_FileOpenDialog, NULL, CLSCTX\_ALL,

IID\_IFileOpenDialog, reinterpret\_cast<void\*\*>(&pFileOpen));

if (SUCCEEDED(hr)) {

hr = pFileOpen->Show(NULL);

if (SUCCEEDED(hr)) {

IShellItem\* pItem;

hr = pFileOpen->GetResult(&pItem);

if (SUCCEEDED(hr)) {

PWSTR pszFilePath;

hr = pItem->GetDisplayName(SIGDN\_FILESYSPATH, &pszFilePath);

if (SUCCEEDED(hr)) {

HANDLE hFile = CreateFile(pszFilePath, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

DWORD fileSize = GetFileSize(hFile, NULL);

LPSTR buffer = (LPSTR)GlobalAlloc(GPTR, fileSize + 1);

DWORD read;

if (ReadFile(hFile, buffer, fileSize, &read, NULL)) {

SetWindowTextA(hWndEdit, buffer);

} else {

MessageBoxA(hwnd, "Cannot read file", "Error", MB\_OK);

}

GlobalFree((HGLOBAL)buffer);

ShowWindow(hWndEdit, SW\_SHOW);

SetFocus(hWndEdit);

CloseHandle(hFile);

}

pItem->Release();

}

}

pFileOpen->Release();

}

}

void SaveFile(HWND hwnd) {

IFileSaveDialog\* pFileSave;

HRESULT hr = CoCreateInstance(CLSID\_FileSaveDialog, NULL, CLSCTX\_ALL,

IID\_IFileSaveDialog, reinterpret\_cast<void\*\*>(&pFileSave));

if (SUCCEEDED(hr)) {

hr = pFileSave->Show(NULL);

if (SUCCEEDED(hr)) {

IShellItem\* pItem;

hr = pFileSave->GetResult(&pItem);

if (SUCCEEDED(hr)) {

PWSTR pszFilePath;

hr = pItem->GetDisplayName(SIGDN\_FILESYSPATH, &pszFilePath);

if (SUCCEEDED(hr)) {

HANDLE hFile = CreateFile(pszFilePath, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, CREATE\_NEW, FILE\_ATTRIBUTE\_NORMAL, NULL);

DWORD fileSize = GetWindowTextLength(hWndEdit) + 1;

LPSTR buffer = (LPSTR)GlobalAlloc(GPTR, fileSize);;

GetWindowTextA(hWndEdit, buffer, fileSize);

DWORD wroted;

if (WriteFile(hFile, (void\*)buffer, fileSize, &wroted, NULL)) {

MessageBox(hwnd, L"File successfully saved", L"Saved", MB\_OK);

CloseHandle(hFile);

}

GlobalFree((HGLOBAL)buffer);

}

}

}

}

}

void FontChoice(HWND hwnd) {

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf;

GetObject(hEditFont, sizeof(LOGFONT), &lf);

cf.Flags = CF\_EFFECTS | CF\_SCREENFONTS | CF\_INITTOLOGFONTSTRUCT;

cf.hwndOwner = hwnd;

cf.lpLogFont = &lf;

if (!ChooseFont(&cf))

return;

HFONT hf = CreateFontIndirect(&lf);

hEditFontColor = cf.rgbColors;

if (hf) {

SendMessage(hWndEdit, WM\_SETFONT, (WPARAM)hf, TRUE);

SendMessage(hWndEdit, WM\_CTLCOLOREDIT, (WPARAM)hf, TRUE);

}

}

void BackgroundColor(HWND hwnd) {

CHOOSECOLOR chColor;

COLORREF acrCustClr[16];

chColor.lStructSize = sizeof(chColor);

chColor.lpCustColors = (LPDWORD)acrCustClr;

chColor.rgbResult = hEditBackgroundColor;

chColor.Flags = CC\_FULLOPEN | CC\_RGBINIT;

chColor.hwndOwner = NULL;

chColor.lpfnHook = Lpcchookproc;

if (ChooseColor(&chColor)) {

hEditBackgroundColor = chColor.rgbResult;

HWND hStatic = GetDlgItem(hwnd, GetDlgCtrlID(hWndEdit));

HDC hdcStatic = GetDC(hStatic);

SendMessage(hWndEdit, WM\_CTLCOLOREDIT, (WPARAM)hdcStatic, (LPARAM)hStatic);

};

}