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

ОТЧЕТ

к лабораторной работе №2

на тему

**РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN32 И GDI**

Студент Е. С. Кахновский

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

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.

1. **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, такие как WM\_PAINT, WM\_KEYDOWN, WM\_TIMER и другие.

– WNDCLASSEX: это структура, используемая для описания класса окна. Включает в себя настройки, такие как стиль, функция обратного вызова (WndProc), иконки, курсоры и т.д. Зарегистрированный класс используется для создания окон.

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

– Создание окна: Сначала регистрируется класс окна с помощью RegisterClassEx, который описывает параметры окна. Затем с помощью CreateWindow создается главное окно приложения.

– 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 разделено на несколько частей с разными цветами, представляющими землю, фон и т.д. Функция DrawAirplane отрисовывает "самолет" с использованием различных элементов: фюзеляж, хвост, крыло, окна и шасси (в зависимости от состояния шасси).

– Управление: Пользователь может управлять "самолетом" с помощью клавиатуры. Стрелки влево и вправо увеличивают/уменьшают скорость "самолета", а стрелки вверх и вниз изменяют его вертикальное положение. Пробел переключает шасси "самолета" между выдвинутым и убранным состоянием.

– Хук клавиатуры: Приложение также демонстрирует использование глобального хука клавиатуры с помощью функций SetWindowsHookEx и UnhookWindowsHookEx. Хук отслеживает нажатие клавиши пробела и выводит сообщение в отладочное окно о состоянии шасси. Таймер: Приложение использует таймер для периодического обновления положения "самолета" в окне и его отрисовки.

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

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной работы было создано оконное приложение, позволяющее управлять модель самолета (взаимодействие происходит по нажатию стрелок на клавиатуре, пробел – открыть/закрыть шасси)(рисунок 1).

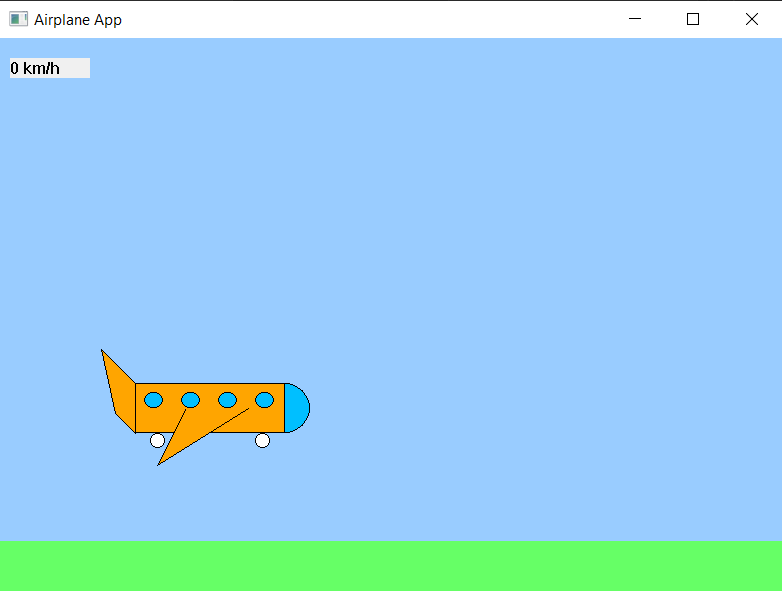


Рисунок *1* ─ Основное окно приложения

Так же присутствует обработка ошибок, таких как, приземление на землю без выпущенных шасси (рисунок 2).

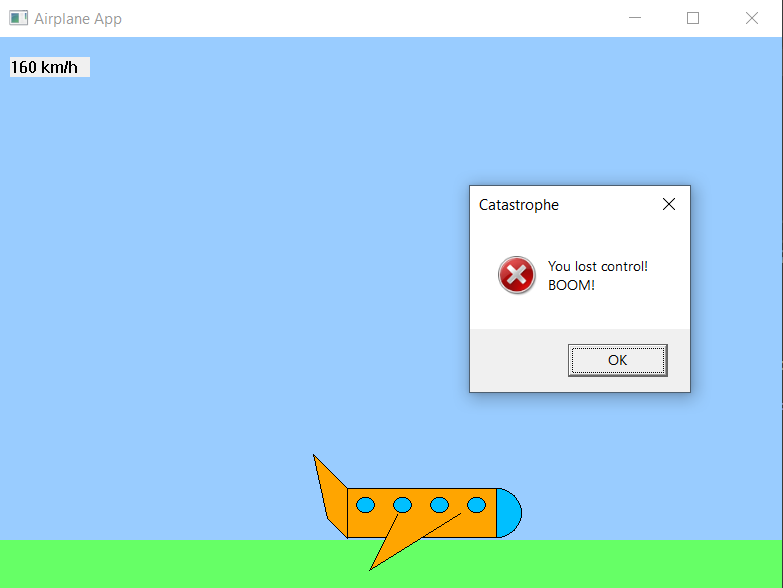


Рисунок *2* ─ Окно потери контроля над самолетом

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было разработано оконное приложение на WIN32 API с использованем расширенного оконного интерфейса Win32 и GDI, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, использование механизма перехвата сообщений (winhook).

# 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.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 ­– Файл RegexChecker.cpp

#include <windows.h>

#include <regex>

#include "RegexCheckerDef.h"

// Global variables

HWND hMainWindow;

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

void CreateUIElements(HINSTANCE hInstance);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

// Initializing the MainWindow

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, IDI\_APPLICATION);

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = L"SimpleApp";

wcex.hIconSm = LoadIcon(wcex.hInstance, IDI\_APPLICATION);

if (!RegisterClassEx(&wcex))

{

MessageBox(NULL, L"Call to RegisterClassEx failed!", L"Error", MB\_ICONERROR);

return 1;

}

// Creating MainWindow

hMainWindow = CreateWindow(L"SimpleApp", L"Regex Checker", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 200, NULL, NULL, hInstance, NULL);

if (!hMainWindow)

{

MessageBox(NULL, L"Call to CreateWindow failed!", L"Error", MB\_ICONERROR);

return 1;

}

// Creating interface elements (fields and buttons)

CreateUIElements(hInstance);

// Showing MainWindow

ShowWindow(hMainWindow, nCmdShow);

UpdateWindow(hMainWindow);

// The main message loop

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

void CreateUIElements(HINSTANCE hInstance)

{

// Creating a label for a regular expression

CreateWindow(L"STATIC", L"Regex: ", WS\_CHILD | WS\_VISIBLE, 10, 20, 50, 20, hMainWindow, (HMENU)ID\_REGEX\_LABEL, hInstance, NULL);

// Creating a editfield for a regular expression

CreateWindow(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER , 65, 20, 200, 20, hMainWindow, (HMENU)ID\_REGEX\_EDIT, hInstance, NULL);

// Creating a label for a text

CreateWindow(L"STATIC", L"Text: ", WS\_CHILD | WS\_VISIBLE, 10, 50, 50, 20, hMainWindow, (HMENU)ID\_REGEX\_LABEL, hInstance, NULL);

// Creating a editfield for a text

CreateWindow(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL, 65, 50, 200, 20, hMainWindow, (HMENU)ID\_TEXT\_EDIT, hInstance, NULL);

// Creating button

CreateWindow(L"BUTTON", L"Check", WS\_CHILD | WS\_VISIBLE, 150, 80, 100, 30, hMainWindow, (HMENU)ID\_CHECK\_BUTTON, hInstance, NULL);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

// Gets HWND for all controls

HWND hRegexField = GetDlgItem(hWnd, ID\_REGEX\_EDIT);

HWND hTextField = GetDlgItem(hWnd, ID\_TEXT\_EDIT);

HWND hButton = GetDlgItem(hWnd, ID\_CHECK\_BUTTON);

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ID\_CHECK\_BUTTON:

if (HIWORD(wParam) == BN\_CLICKED)

{

// Getting a regular expression from hRegexField

WCHAR regexBuffer[256];

GetWindowText(hRegexField, regexBuffer, sizeof(regexBuffer) / sizeof(regexBuffer[0]));

// Getting a text from hTextField

WCHAR textBuffer[256];

GetWindowText(hTextField, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

// Converting a regular expression to an std::regex object

std::wregex regex;

try

{

regex = std::wregex(regexBuffer);

}

catch (const std::regex\_error&)

{

MessageBox(hWnd, L"Invalid regular expression!", L"Error", MB\_ICONERROR);

break;

}

// Checking whether the text matches the regular expression

// Output of the result in MessageBox

if (std::regex\_match(textBuffer, regex))

{

MessageBox(hWnd, L"Success", L"Result", MB\_ICONINFORMATION);

}

else

{

MessageBox(hWnd, L"Failure", L"Result", MB\_ICONERROR);

}

}

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

break;

}

return 0;

}

Листинг 2 ­– Файл RegexCheckerDef.h

#pragma once

// Identifiers of interface elements

// Buttons 1\*

#define ID\_CHECK\_BUTTON 11

// Labels 2\*

#define ID\_REGEX\_LABEL 21

#define ID\_TEXT\_LABEL 22

// Edits 3\*

#define ID\_REGEX\_EDIT 31

#define ID\_TEXT\_EDIT 32