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 №1
на тему

ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API. ОКОННОЕ ПРИЛОЖЕНИЕ WIN 32 С МИНИМАЛЬНОЙ ДОСТАТОЧНОЙ ФУНКЦИОНАЛЬНОСТЬЮ. ОБРАБОТКА ОСНОВНЫХ ОКОННЫХ СООБЩЕНИЙ.

Студент М. А. Шкарубский
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью выполнения данной лабораторной работы является изучение основных принципов работы с Win32 API, обработка основных оконных сообщений (создание и удаление окна, сообщения управляющих элементов), разработка оконного приложения с минимальной функциональной достаточностью – приложение для чтения и отображения файлов в формате CSV.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В ОС Windows реализована объектно-ориентированная идеология. Базовый объект системы – окно, поведение которого определяется методом, называемым функцией окна. Графический образ окна на экране дисплея – прямоугольная рабочая область.

Независимо от своего типа любой объект Windows идентифицируется описателем или дескриптором (handle). Дескриптор – это ссылка на объект. Все взаимоотношения программного кода с объектом осуществляются только через его дескриптор.

Интерфейс прикладного программирования (API – Application Programming Interface) представляет собой совокупность 32-битных функций (Win32 API), которые предназначены для создания приложений (программ), работающих под управлением Microsoft Windows. Функции объявлены в заголовочных файлах. Главный из них − файл windows.h, в котором содержатся ссылки на другие заголовочные файлы [1].

Окно – это прямоугольная область экрана, в котором приложение отображает информацию и получает реакцию от пользователя. Одновременно на экране может отображаться несколько окон, в том числе, окон других приложений, однако лишь одно из них может получать реакцию от пользователя – активное окно. Пользователь использует клавиатуру, мышь и прочие устройства ввода для взаимодействия с приложением, которому принадлежит активное окно.

Каждое 32-битное приложение создает, по крайней мере, одно окно, называемое главным окном, которое обеспечивает пользователя основным интерфейсом взаимодействия с приложением. Кроме главного окна, приложение может использовать еще и другие типы окон: управляющие элементы, диалоговые окна, окна-сообщения.

Управляющий элемент – окно, непосредственно обеспечивающее тот или иной способ ввода информации пользователем. К управляющим элементам относятся: кнопки, поля ввода, списки, полосы прокрутки и т.п. Управляющие элементы обычно находятся в каком-либо диалоговом окне.

Диалоговое окно – это временное окно, содержащее управляющие элементы, обычно использующееся для получения дополнительной информации от пользователя. Диалоговые окна бывают модальные и немодальные. Модальное диалоговое окно требует, чтобы пользователь обязательно ввел обозначенную в окне информацию и закрыл окно прежде, чем приложение продолжит работу. Немодальное диалоговое окно позволяет пользователю, не закрывая диалогового окна, переключаться на другие окна этого приложения.

Окно-сообщение – это диалоговое окно предопределенного системой формата, предназначенное для вывода небольшого текстового сообщения с одной или несколькими кнопками.

В отличие от традиционного программирования на основе линейных алгоритмов, программы для Windows строятся по принципам событийно-управляемого программирования – стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна. С каждым окном в Windows связана определенная функция обработки событий. События для окон называются сообщениями. Сообщение относится к тому или иному типу, идентифицируемому 32-битным целым числом (например, WM_COMMAND, WM_CREATE и WM_DESTROY), и сопровождается парой 32-битных параметров (WPARAM и LPARAM), интерпретация которых зависит от типа сообщения [3].

Каждое окно принадлежит определенному классу окон. Окна одного класса имеют схожий вид, обслуживаются общей процедурой обработки событий, имеют одинаковые иконки и меню. Обычно каждое приложение создает для главного окна программы свой класс. Если приложению требуются дополнительные нестандартные окна, оно регистрирует другие классы. Стандартные диалоги и управляющие элементы принадлежат к предопределенным классам окон, для них не надо регистрировать новые классы.

Управляющие элементы, как и другие окна, принадлежат тому или иному классу окон. Windows предоставляет несколько предопределенных классов управляющих элементов. Программа может создавать управляющие элементы поштучно при помощи функции CreateWindow или оптом, загружая их вместе с шаблоном диалога из своих ресурсов. Управляющие элементы – это всегда дочерние окна. Управляющие элементы при возникновении некоторых событий, связанных с реакцией пользователя, посылают своему родительскому окну сообщения-оповещения.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В результате выполнения лабораторной работы было создано приложение для считывания CSV файлов и их отображения (см. рисунок 1).

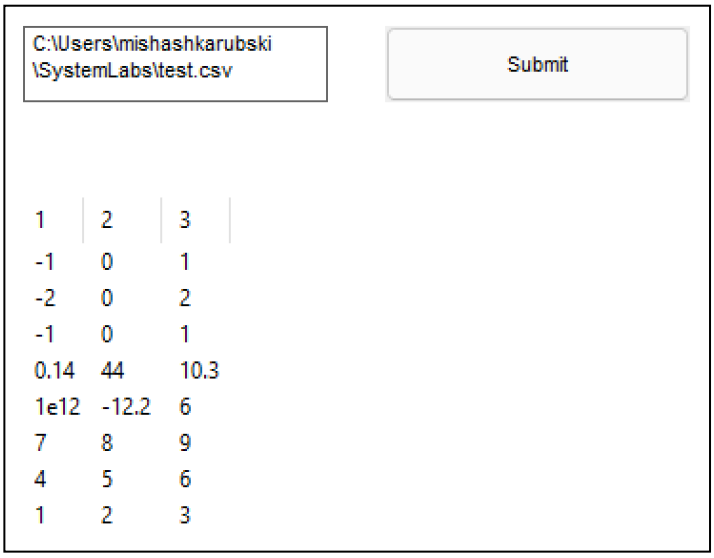

Рисунок 1 – Client area приложения

По нажатию кнопки Submit происходит конвертация строки, введенной пользователем, в путь, по которому открывается CSV файл. Дальнейшие действия включают в себя вычисление размеров (в частности ширины) столбцов таблицы и отображение данных.

В приложении предусмотрено предупреждение пользователя об ошибочном вводе (см. рисунок 2).

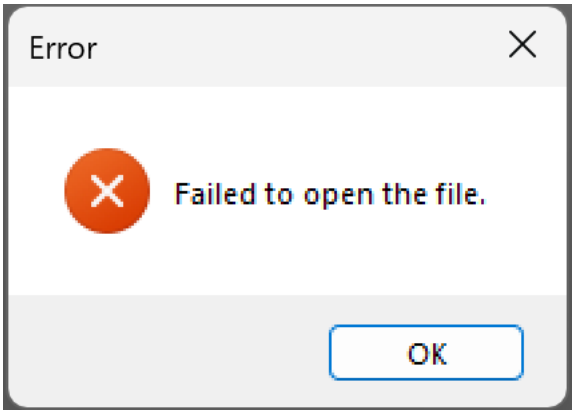

Рисунок 2 – Результат работы при неверном пользовательском вводе

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные принципы работы с Win32 API: виды окон, классы окон и их регистрация, обработка сообщений разных типов. Было создано оконное приложение с минимальной функциональной достаточностью – приложение для считывания CSV файлов с возможностью простейшей визуализации. Была предусмотрена обработка ввода неверных значений пользователем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Справочник по программированию для API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/.

[2] Безруков В.А. Win32 API. Программирование / учебное пособие. – СПб: СПбГУ ИТМО, 2009. – 90 с.

[3] Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: https://dims.karelia.ru/win32/.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл Lab1.cpp

```
#pragma comment(linker,"\"/manifestdependency:type='win32' \
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \
processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'\"")

#ifndef UNICODE
#define UNICODE
#endif
#define DEFAULT_WIDTH 1200
#define DEFAULT_HEIGHT 900

#include <iostream>
#include <windows.h>
#include <commctrl.h>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>


HWND gListView;
HWND pathInputHandle;
HWND pathButtonHandle;
HWND xInputHandle;
HWND yInputHandle;
HWND graphButtonHandle;
HWND graphHandle;

struct Point {
    int x;
    int y;
};

std::vector<Point> points;
wchar_t csvFilePath[128];

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam);
std::vector<std::wstring> GetCSVColumns(const std::wstring& filePath);
std::vector<std::wstring> ReadCSVFileLine(const std::wstring& line, wchar_t
delimiter);
void DisplayCSVInListView(const std::wstring& filePath, HWND listViewHandle);
void LinePlot(HWND hwnd);


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR
lpCmdLine, int nCmdShow) {
    const wchar_t CLASS_NAME[] = L"Data visualizer";

    WNDCLASS wc = { 0 };
```

```
wc.lpfnWndProc = WindowProc;
wc.hInstance = hInstance;
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
wc.lpszClassName = CLASS_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(
    0,
    CLASS_NAME,
    L"Data vizualizer - Main window",
    WS_OVERLAPPEDWINDOW,

    // x, y, width, height
    CW_USEDEFAULT,  CW_USEDEFAULT, DEFAULT_WIDTH, DEFAULT_HEIGHT,

    nullptr, // parent
    nullptr, // menu
    hInstance, // instance handle
    nullptr // additional data
);

pathInputHandle = CreateWindowEx(
    0, L"EDIT", L"Path to .csv file",
    WS_VISIBLE | WS_CHILD | WS_VISIBLE | WS_BORDER | ES_MULTILINE,
    10, 10, 160, 40, hwnd, nullptr, hInstance, nullptr
);

pathButtonHandle = CreateWindowEx(
    0, L"BUTTON", L"Submit",
    WS_VISIBLE | WS_CHILD,
    200, 10, 160, 40, hwnd, nullptr, hInstance, nullptr
);

xInputHandle = CreateWindowEx(
    0, L"EDIT", L"X",
    WS_VISIBLE | WS_CHILD | WS_VISIBLE | WS_BORDER,
    400, 520, 80, 40, hwnd, nullptr, hInstance, nullptr
);

yInputHandle = CreateWindowEx(
    0, L"EDIT", L"Y",
    WS_VISIBLE | WS_CHILD | WS_VISIBLE | WS_BORDER,
    500, 520, 80, 40, hwnd, nullptr, hInstance, nullptr
);

graphButtonHandle = CreateWindowEx(
    0, L"BUTTON", L"Add to graph",
    WS_VISIBLE | WS_CHILD,
    600, 520, 160, 40, hwnd, nullptr, hInstance, nullptr
);

graphHandle = CreateWindowEx(
    0, L"STATIC", L"Lineplot",
    WS_VISIBLE | WS_CHILD | SS_BLACKRECT,
    400, 100, 600, 400, hwnd, nullptr, hInstance, nullptr
);
```

```
    HFONT font = CreateFont(14, 0, 0, 0, FW_NORMAL, FALSE, FALSE, FALSE,
DEFAULT_CHARSET, OUT_OUTLINE_PRECIS,
        CLIP_DEFAULT_PRECIS, CLEARTYPE_QUALITY, VARIABLE_PITCH,
TEXT("Arial"));

    SendMessage(pathInputHandle, WM_SETFONT, (WPARAM) font, TRUE);
    SendMessage(pathButtonHandle, WM_SETFONT, (WPARAM) font, TRUE);
    SendMessage(xInputHandle, WM_SETFONT, (WPARAM) font, TRUE);
    SendMessage(yInputHandle, WM_SETFONT, (WPARAM) font, TRUE);
    SendMessage(graphButtonHandle, WM_SETFONT, (WPARAM) font, TRUE);

    if (hwnd == nullptr)
        return 1;

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    MSG msg = { };

    while (GetMessage(&msg, NULL, 0, 0) > 0) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam) {
    switch (uMsg) {
        case WM_COMMAND:
        {
            if (HIWORD(wParam) == BN_CLICKED) {
                if ((HWND)lParam == graphButtonHandle) {
                    wchar_t xText[128];
                    wchar_t yText[128];

                    GetWindowText(xInputHandle, xText, 128);
                    GetWindowText(yInputHandle, yText, 128);

                    int xValue = _wtoi(xText);
                    int yValue = _wtoi(yText);

                    if ((std::strcmp((const char*) xText, "0") && xValue ==
0) || (std::strcmp((const char*)yText, "0") && yValue == 0)) {
                        MessageBox(nullptr, L"Ng wha?", L"ANOGA", MB_OK |
MB_ICONERROR);
                        break;
                    }

                    points.push_back({ xValue, yValue });

                    // Update the graph
                    LinePlot(graphHandle);
                }

                else if ((HWND)lParam == pathButtonHandle) {
                    GetWindowText(pathInputHandle, csvFilePath, 128);
```

```
                    gListView = CreateWindow(
                        WC_LISTVIEW, L"",
                        WS_VISIBLE | WS_CHILD | LVS_REPORT | LVS_EDITLABELS,
                        10, 100, 400, 300, hwnd, nullptr, nullptr, nullptr
                    );

                    std::vector<std::wstring> headers =
GetCSVColumns(csvFilePath);

                    if (!headers.size())
                        return 0;

                    LV_COLUMN lvColumn;
                    lvColumn.mask = LVCF_TEXT;

                    for (int i = 0; i < headers.size(); ++i) {
                        lvColumn.pszText = headers[i].data();
                        ListView_InsertColumn(gListView, i, &lvColumn);
                    }

                    DisplayCSVInListView(csvFilePath, gListView);
                }
            }
        };

            break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }

    return 0;
}

std::vector<std::wstring> GetCSVColumns(const std::wstring& filePath) {
    std::vector<std::wstring> headers;
    std::wstring line;
    std::wifstream file(filePath);

    if (!file) {
        MessageBox(nullptr, L"Ng wha?", L"ANOGA", MB_OK | MB_ICONERROR);
        return headers;
    }

    std::getline(file, line);
    headers = ReadCSVFileLine(line, L',');
    //line.replace(line.find(line), line.length(), L"");
    //file << line << std::endl;
    //file.close();

    file.close();

    return headers;
```

```
}


std::vector<std::wstring> ReadCSVFileLine(const std::wstring& line, wchar_t
delimiter) {
    std::vector<std::wstring> tokens;
    std::wstringstream ss(line);
    std::wstring token;

    while (std::getline(ss, token, delimiter))
        tokens.push_back(token);

    return tokens;
}


void DisplayCSVInListView(const std::wstring& filePath, HWND listViewHandle)
{
    ListView_DeleteAllItems(listViewHandle);

    std::wifstream file(filePath, std::ios::in | std::ios::binary);

    if (!file) {
        MessageBox(nullptr, L"Ng wha?", L"ANOGA", MB_OK | MB_ICONERROR);
        return;
    }

    std::wstring line;
    while (std::getline(file, line)) {
        std::vector<std::wstring> row = ReadCSVFileLine(line, L',');

        LVITEM lvItem;
        lvItem.mask = LVIF_TEXT;
        lvItem.iItem = 0;
        lvItem.iSubItem = 0;
        lvItem.pszText = (LPWSTR) row[0].c_str();

        int itemIndex = ListView_InsertItem(listViewHandle, &lvItem);

        for (size_t i = 1; i < row.size(); i++) {
            lvItem.mask = LVIF_TEXT;
            lvItem.iItem = itemIndex;
            lvItem.iSubItem = i;
            lvItem.pszText = (LPWSTR) row[i].c_str();

            ListView_SetItem(gListView, &lvItem);
        }
    }

    for (int i = 0; i < ListView_GetItemCount(listViewHandle); ++i)
        ListView_SetColumnWidth(listViewHandle, i, LVSCW_AUTOSIZE_USEHEADER |
LVTVIF_FIXEDWIDTH);

    file.close();
}

void LinePlot(HWND hwnd) {
    HWND graphHandle = GetDlgItem(hwnd, 4);
```

```
    RECT rect;

    GetClientRect(hwnd, &rect);

    HDC hdc = GetDC(hwnd);
    HPEN pen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
    HBRUSH brush = CreateSolidBrush(RGB(245, 245, 245));

    SelectObject(hdc, pen);
    SelectObject(hdc, brush);
    FillRect(hdc, &rect, brush);

    Point prevPoint;
    bool isFirstPoint = true;

    int ySpan = (rect.bottom + rect.top) / 2;
    int xSpan = (rect.left + rect.right) / 2;

    for (int i = 0; i < points.size(); ++i) {
        Point point = points[i];

        SelectObject(hdc, brush);

        if ((rect.bottom + rect.top) / 2 - point.y < rect.top || (rect.left +
rect.right) / 2 + point.x > rect.right) {
            points.erase(points.begin() + i);
            continue;
        }


        Ellipse(
            hdc,
            xSpan + point.x - 4, ySpan - point.y - 4,
            xSpan + point.x + 4, ySpan - point.y + 4
        );

        if (isFirstPoint) {
            prevPoint.x = point.x;
            prevPoint.y = point.y;
            isFirstPoint = false;
        } else {
            MoveToEx(hdc, xSpan + prevPoint.x, ySpan - prevPoint.y, nullptr);
            LineTo(hdc, xSpan + point.x, ySpan - point.y);
            prevPoint.x = point.x;
            prevPoint.y = point.y;
        }
    }

    ReleaseDC(graphHandle, hdc);
    DeleteObject(brush);
}
```