МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВПО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Работа с Windows API

Отчет по лабораторной работе №3 дисциплины

«Технологии программирования»

Выполнил студент группы ИВТб-21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Седов М.Д./

Проверил доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2019

**Цель работы**: получить базовые навыки работы с набором функций для создания оконных приложения при помощи Windows API

**Задание:**

1) Написать программу, главное родительское окно, которое содержит так же два дочерних окна.

2) На главное окне находятся две кнопки, меню с двумя полями.

3) Первое поле меню отвечает за ввод данных.

4) Второе поле меню отвечает за выполнение операций.

5) Реализовать нахождение минимального пути в графе, и узнать, достижим ли проход графа из точки А в точку Б.

**Исходный текст программы:**

#include <Windows.h>

#include <limits.h>

#define ButtonFirst 0

#define ButtonTo 1

#define MenuFirst 2

#define MenuTo 3

#define CmbBox 4

#define ChildWindow1 5

#define ChildWindow2 6

#define HOTKEX 7

bool onRead = false, onSolve = false;

char Buffer[102] = { 0 };

HWND hwnd2, hwnd3;

bool read\_graph();

void minDistance();

void inspectionWay();

LRESULT CALLBACK ChildProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message)

{

case WM\_CLOSE:

ShowWindow(hwnd, SW\_HIDE);

break;

default:

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

}

}

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

{

switch (message) {

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case ButtonFirst:

minDistance();

break;

case ButtonTo:

inspectionWay();

break;

case MenuFirst:

ShowWindow(hwnd2, SW\_NORMAL);

UpdateWindow(hwnd2);

if (read\_graph())

onSolve = true;

break;

case MenuTo:

if (onSolve){

onRead = true;

ShowWindow(hwnd3, SW\_NORMAL);

UpdateWindow(hwnd3);

}

else

MessageBox(NULL, (LPCWSTR)L"Not data for solves", (LPCWSTR)L"Attention", MB\_OK);

break;

default:

break;

}

break;

case WM\_DESTROY: PostQuitMessage(0); break;

default: return DefWindowProc(hwnd, message, wParam, lParam); break;

}

}

HWND CreateButton(LPWSTR text, int x, int y, int width, int height, HWND hwnd, int hmenu) {

return CreateWindow(

L"BUTTON", // Predefined class; Unicode assumed

text, // Button text

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | BS\_PUSHBUTTON, // Styles

x, // x position

y, // y position

width, // Button width

height, // Button height

hwnd, // Parent window

(HMENU)hmenu,

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

NULL

);

}

HWND CreateComboBox(LPWSTR text, int x, int y, int width, int height, HWND hwnd, int hmenu) {

return CreateWindow(

L"COMBOBOX",

TEXT(""),

CBS\_DROPDOWN | CBS\_HASSTRINGS | WS\_CHILD | WS\_OVERLAPPED | WS\_VISIBLE,

x,

y,

width,

height,

hwnd,

(HMENU)hmenu,

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

NULL

);

}

WNDCLASS myRegistryClass(HINSTANCE hInst, int cmdMode, WNDPROC proc, LPCWSTR s) {

WNDCLASS wc{sizeof(WNDCLASS)};

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = HBRUSH(BLACK\_BRUSH);

wc.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wc.hIcon = LoadIcon(nullptr, IDI\_APPLICATION);

wc.hInstance = hInst;

wc.lpfnWndProc = proc;

wc.lpszClassName = s;

wc.lpszMenuName = nullptr;

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

RegisterClass(&wc);

return wc;

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstant, PWSTR pCmdLine, int nCmdShow)

{

MSG msg{};

WNDCLASS wc = myRegistryClass(hInstance, nCmdShow, WndProc, L"MyAppClass");

HWND hwnd = CreateWindow(

wc.lpszClassName,

L"Главное окно",

WS\_OVERLAPPEDWINDOW ^ WS\_THICKFRAME,

0,

0,

600,

600,

nullptr,

nullptr,

wc.hInstance,

nullptr

);

wc = myRegistryClass(hInstance, nCmdShow, ChildProc, L"MyChildClass");

hwnd2 = CreateWindow(

wc.lpszClassName,

L"Ввод данных",

WS\_CHILD | WS\_OVERLAPPEDWINDOW,

300,

0,

200,

100,

hwnd,

HMENU(ChildWindow1),

wc.hInstance,

nullptr

);

wc = myRegistryClass(hInstance, nCmdShow, ChildProc, L"MyChild2Class");

hwnd3 = CreateWindow(

wc.lpszClassName,

L"Выполнение действий",

WS\_CHILD | WS\_OVERLAPPEDWINDOW,

300,

400,

200,

100,

hwnd,

HMENU(ChildWindow1),

wc.hInstance,

nullptr

);

if (hwnd == INVALID\_HANDLE\_VALUE)

return EXIT\_FAILURE;

HMENU hMenu = CreateMenu();

HMENU hMen = CreatePopupMenu();

AppendMenu(hMenu, MF\_STRING | MF\_POPUP, (UINT)hMen, L"New");

AppendMenu(hMen, MF\_STRING, 2, L"First");

AppendMenu(hMen, MF\_STRING, 3, L"To");

SetMenu(hwnd, hMenu);

HWND hwndButton1 = CreateButton(L"1", 50, 50, 75, 50, hwnd, ButtonFirst);

HWND hwndButton2 = CreateButton(L"2", 150, 50, 75, 50, hwnd, ButtonTo);

//HWND hwndComboBox = CreateComboBox(L"", 20, 200, 50, 10, hwnd2, CmbBox);

EnableWindow(hwndButton1, false);

EnableWindow(hwndButton2, false);

ShowWindow(hwnd, SW\_SHOWMAXIMIZED);

UpdateWindow(hwnd);

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

{

TranslateMessage(&msg);

if (onRead) {

EnableWindow(hwndButton1, true);

EnableWindow(hwndButton2, true);

}

DispatchMessage(&msg);

}

return msg.wParam;

}

DWORD g\_BytesTransferred = 0;

VOID CALLBACK FileIOCompletionRoutine(

\_\_in DWORD dwErrorCode,

\_\_in DWORD dwNumberOfBytesTransfered,

\_\_in LPOVERLAPPED lpOverlapped)

{

g\_BytesTransferred = dwNumberOfBytesTransfered;

}

bool read\_graph() {

OPENFILENAME ofn; // common dialog box structure

TCHAR szFile[260] = { 0 }; // if using TCHAR macros

HANDLE hf;

DWORD dwBytesRead = 0;

OVERLAPPED ol = { 0 };

// Initialize OPENFILENAME

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd2;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"Text\0\*.TXT\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetOpenFileName(&ofn) == TRUE)

{

hf = CreateFile(ofn.lpstrFile, GENERIC\_READ,

0, (LPSECURITY\_ATTRIBUTES)NULL,

OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL,

(HANDLE)NULL);

if (hf == INVALID\_HANDLE\_VALUE) return false;

if (FALSE == ReadFileEx(hf, Buffer, 101, &ol, FileIOCompletionRoutine)) return false;

SleepEx(5000, TRUE);

DWORD dwBytesToRead = (DWORD)strlen(Buffer);

CloseHandle(hf);

}

return true;

}

void minDistance() {

/////////////////////////////// select mindistance on graph

int count = Buffer[0] - 48;

int \*graph = (int \*)malloc(count \* count \* sizeof(int));

int \*W = (int \*)malloc(count \* count \* sizeof(int));

int j = 0;

for (int i = 1; i < strlen(Buffer); i++) {

if (Buffer[i] >= 48 && Buffer[i] <= 57) {

graph[j] = Buffer[i] - 48;

j = j + 1;

}

}

for (int i = 0; i < count \* count; i++)

if (graph[i] != 0)

W[i] = graph[i];

else

W[i] = 100;

for (int k = 0; k < count; k++)

for (int i = 0; i < count; i++)

for (int j = 0; j < count; j++)

if (W[i \* count + j] != 0)

W[i \* count + j] = min(W[i \* count + j], W[i \* count + k] + W[k \* count + j]);

//////////////////////////////

////////////////////////////// create buffer for write

int size = (count \* count + (count - 1) \* 2 + (count - 1) \* count + 1);

WCHAR WriteBuffer[MAX\_PATH] = L"";

strcpy(Buffer, Buffer + 3);

for (int i = 0, j = 0; i < strlen(Buffer); i++) {

if (Buffer[i] >= 48 && Buffer[i] <= 57) {

if (W[j] == 100)

Buffer[i] = char(48);

else

Buffer[i] = char(W[j] + 48);

j = j + 1;

}

}

/////////////////////////////

OPENFILENAME ofn; // common dialog box structure

HANDLE hfile;

TCHAR szFile[260] = { 0 }; // if using TCHAR macros

DWORD dwBytesToWrite = (DWORD)size;

DWORD dwBytesWritten = 0;

BOOL bErrorFlag = FALSE;

// Initialize SAVEFILENAME

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd2;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"Text\0\*.TXT\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetSaveFileName(&ofn) == TRUE) {

hfile = CreateFile(ofn.lpstrFile, // name of the write

GENERIC\_WRITE, // open for writing

0, // do not share

NULL, // default security

OPEN\_EXISTING, // create new file only

FILE\_ATTRIBUTE\_NORMAL, // normal file

NULL); // no attr. template

if (hfile == INVALID\_HANDLE\_VALUE) return;

bErrorFlag = WriteFile(

hfile, // open file handle

Buffer, // start of data to write

dwBytesToWrite, // number of bytes to write

&dwBytesWritten, // number of bytes that were written

NULL); // no overlapped structure

if (FALSE == bErrorFlag) return;

CloseHandle(hfile);

free(graph);

free(W);

}

}

void inspectionWay() {

/////////////////////////////// select mindistance on graph

int count = Buffer[0] - 48;

int A = Buffer[3] - 48;

int B = Buffer[5] - 48;

int \*graph = (int \*)malloc(count \* count \* sizeof(int));

int \*W = (int \*)malloc(count \* count \* sizeof(int));

int j = 0;

for (int i = 8; i < strlen(Buffer); i++) {

if (Buffer[i] >= 48 && Buffer[i] <= 57) {

graph[j] = Buffer[i] - 48;

j = j + 1;

}

}

for (int i = 0; i < count \* count; i++)

if (graph[i] != 0)

W[i] = graph[i];

else

W[i] = 100;

for (int k = 0; k < count; k++)

for (int i = 0; i < count; i++)

for (int j = 0; j < count; j++)

if (W[i \* count + j] != 0)

W[i \* count + j] = min(W[i \* count + j], W[i \* count + k] + W[k \* count + j]);

//////////////////////////////

////////////////////////////// create buffer for write

int size = (count \* count + (count - 1) \* 2 + (count - 1) \* count + 1);

WCHAR WriteBuffer[MAX\_PATH] = L"";

if (W[A \* count + B] == 0)

WriteBuffer[0] = 'N';

else

WriteBuffer[0] = 'Y';

/////////////////////////////

OPENFILENAME ofn; // common dialog box structure

HANDLE hfile;

TCHAR szFile[260] = { 0 }; // if using TCHAR macros

DWORD dwBytesToWrite = (DWORD)size;

DWORD dwBytesWritten = 0;

BOOL bErrorFlag = FALSE;

// Initialize SAVEFILENAME

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd2;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"Text\0\*.TXT\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

if (GetSaveFileName(&ofn) == TRUE) {

hfile = CreateFile(ofn.lpstrFile, // name of the write

GENERIC\_WRITE, // open for writing

0, // do not share

NULL, // default security

OPEN\_EXISTING, // create new file only

FILE\_ATTRIBUTE\_NORMAL, // normal file

NULL); // no attr. template

if (hfile == INVALID\_HANDLE\_VALUE) return;

bErrorFlag = WriteFile(

hfile, // open file handle

WriteBuffer, // start of data to write

dwBytesToWrite, // number of bytes to write

&dwBytesWritten, // number of bytes that were written

NULL); // no overlapped structure

if (FALSE == bErrorFlag) return;

CloseHandle(hfile);

free(graph);

free(W);

}

}

**Вывод**

В ходе данной лабораторной работы были освоены базовые возможности для работы с Windows API. Так же были получены навыки работы с родительскими и дочерними окнами, в котором находились кнопки с соответствующими функция, вызываемыми при нажатии на них, меню с выбором определенного поля. Были реализованы диалоговые окна для сохранения и открытия в/из файл/а. Из файла считывался граф в виде матрицы с весами между вершинами.