МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №4

Тема: Создание и использование DLL

Курс: Системное программирование

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Филипишин Д.А.

Донецк – 2021

**Задание**

Задание. Создать две DLL.

Первая экспортирует функцию рисования графика из лабораторной работы №3. Аргументы функции: дескриптор окна, дескриптор контекста устройства, значения Xmin и Xmax.

Вторая DLL экспортирует функцию вывода пользовательского блока диалога из лабораторной работы №3. На вход ей подаются значения Xmin и Xmax. Функция должна возвращать те же параметры со значениями, введенными пользователем.

Модифицировать лабораторную работу №3 таким образом, чтобы функции рисования графика и вывода пользовательского блока диалога вызывались из DLL. Первая DLL должна загружаться явно (с помощью функций LoadLibrary() или LoadLibraryEx()) при создании главного окна приложения, вторая – во время первого обращения к блоку диалога. Если первую DLL не удаётся загрузить (или не удаётся получить адрес нужной функции из DLL), следует закрыть приложение. Если не удаётся загрузить вторую DLL (или не удаётся получить адрес нужной функции из DLL), то заблокировать пункт меню, взывающий блок диалога.

**Листинг**

**DrawGraphic.cpp**

#include "pch.h"

#include "DrawGraphic.h"

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

static double xmin, xmax;

static double miny, maxy;

struct WindowSize {

int width;

int height;

};

WindowSize GetWindowSize(HWND window\_handle) {

RECT window\_rect;

GetClientRect(window\_handle, &window\_rect);

WindowSize result;

result.width = window\_rect.right - window\_rect.left;

result.height = window\_rect.bottom - window\_rect.top;

return result;

}

int DrawAxisX(HDC hdc, WindowSize ws) {

int zeroy;

int textPadding;

TEXTMETRIC tm;

GetTextMetrics(hdc, &tm);

int letterHeight = tm.tmHeight;

int letterWidth = tm.tmAveCharWidth;

if (maxy <= 0) {

textPadding = 10;

zeroy = 7;

MoveToEx(hdc, 0, zeroy, nullptr);

LineTo(hdc, ws.width, zeroy);

}

else if (miny >= 0) {

textPadding = -1 \* letterHeight;

zeroy = ws.height - 7;

MoveToEx(hdc, 0, zeroy, nullptr);

LineTo(hdc, ws.width, zeroy);

}

else {

textPadding = 10;

zeroy = -miny \* ws.height / (maxy - miny);

MoveToEx(hdc, 0, ws.height - zeroy, nullptr);

LineTo(hdc, ws.width, ws.height - zeroy);

zeroy = ws.height - zeroy;

}

double currentX = xmin;

int linePadding = 7;

double dx = (xmax - xmin) / ws.width;

int step = letterWidth \* 12 + linePadding;

for (int i = 0; i < ws.width; i += step) {

std::wstring sx = std::to\_wstring(currentX);

TextOut(hdc, i, zeroy + textPadding, sx.c\_str(), sx.size());

MoveToEx(hdc, i, zeroy - linePadding, nullptr);

LineTo(hdc, i, zeroy + linePadding);

currentX += step \* dx;

}

return zeroy;

}

void DrawAxisY(HDC hdc, WindowSize ws, int zeroy) {

int zerox;

int textPadding;

TEXTMETRIC tm;

GetTextMetrics(hdc, &tm);

int letterHeight = tm.tmHeight;

int letterWidth = tm.tmAveCharWidth;

int voidPadding = 2 \* letterHeight;

if (xmax <= 0) {

textPadding = -10 \* letterWidth;

zerox = ws.width - 7;

MoveToEx(hdc, zerox, 0, nullptr);

LineTo(hdc, zerox, ws.height);

}

else if (xmin >= 0) {

textPadding = 10;

zerox = 7;

MoveToEx(hdc, zerox, 0, nullptr);

LineTo(hdc, zerox, ws.height);

}

else {

textPadding = 10;

zerox = -xmin \* ws.width / (xmax - xmin);

MoveToEx(hdc, zerox, 0, nullptr);

LineTo(hdc, zerox, ws.height);

}

if (miny >= 0) {

zeroy -= voidPadding;

}

double currentY = miny;

int linePadding = 7;

double dy = (maxy - miny) / static\_cast<double>(ws.height);

int step = letterHeight + linePadding;

for (int i = ws.height; i >= 0; i -= step) {

std::wstring sy = std::to\_wstring(currentY);

if (!(zeroy < i && i < zeroy + voidPadding)) {

TextOut(hdc, zerox + textPadding, i - letterHeight / 2, sy.c\_str(), sy.size());

}

MoveToEx(hdc, zerox - linePadding, i, nullptr);

LineTo(hdc, zerox + linePadding, i);

currentY += step \* dy;

}

}

int TransformValue(double x, int newSize, double min, double max) {

int result = newSize - (x - min) / (max - min) \* newSize;

return result;

}

std::vector<int> TransformValues(std::vector<double> oldValues, WindowSize ws) {

std::vector<int> result{};

for (double oldy : oldValues) {

int newy = TransformValue(oldy, ws.height, miny, maxy);

result.push\_back(newy);

}

return result;

}

double CalculateFunc(double x) {

return sin(x) + cos(x);

}

std::vector<double> CalculateFuncValues(double xmin, double xmax, int steps) {

std::vector<double> result{};

double x = xmin;

for (int i = 0; i < steps; ++i, x += (xmax - xmin) / steps) {

result.push\_back(CalculateFunc(x));

}

return result;

}

void DrawFunc(HDC hdc, WindowSize ws) {

auto funcValues = CalculateFuncValues(xmin, xmax, ws.width);

miny = \*std::min\_element(funcValues.begin(), funcValues.end());

maxy = \*std::max\_element(funcValues.begin(), funcValues.end());

auto values = TransformValues(funcValues, ws);

MoveToEx(hdc, 0, values[0], nullptr);

for (int i = 1; i < values.size(); ++i) {

LineTo(hdc, i, values[i]);

}

}

void drawGraphic(HWND hWnd, HDC hdc, double exmin, double exmax) {

xmin = exmin;

xmax = exmax;

WindowSize ws = GetWindowSize(hWnd);

DrawFunc(hdc, ws);

int zeroy = DrawAxisX(hdc, ws);

DrawAxisY(hdc, ws, zeroy);

//LineTo(hdc, 100, 100);

}

**DrawGraphic.h**

#pragma once

#ifdef DRAWGRAPHIC\_EXPORT

#define DRAWGRAPHIC\_API \_\_declspec(dllexport)

#else

#define DRAWGRAPHIC\_API \_\_declspec(dllimport)

#endif

#include <Windows.h>

extern "C" DRAWGRAPHIC\_API void drawGraphic(HWND hwnd, HDC hdc, double xmin, double xmax);

**Graphics\_lab.cpp**

#include <iostream>

#include <vector>

#include <tchar.h>

#include <windows.h>

#include "resource1.h"

#include "utility.hpp"

#include "PaintGuard.hpp"

#include <algorithm>

#include <iterator>

#include <array>

LRESULT CALLBACK MainWndProc(HWND, UINT, WPARAM, LPARAM);

LRESULT CALLBACK ParFun(HWND, UINT, WPARAM, LPARAM);

//std::vector<double> CalculateFuncValues(double xmin, double xmax, int steps);

//double CalculateFunc(double x);

//int TransformValue(double x, int newSize, double min, double max);

struct WindowSize {

int width;

int height;

};

struct xpair {

int xmin;

int xmax;

};

WindowSize GetWindowSize(HWND window\_handle) {

RECT window\_rect;

GetClientRect(window\_handle, &window\_rect);

WindowSize result;

result.width = window\_rect.right - window\_rect.left;

result.height = window\_rect.bottom - window\_rect.top;

return result;

}

//int DrawAxisX(HDC, WindowSize);

//void DrawAxisY(HDC, WindowSize, int);

//void DrawFunc(HDC, WindowSize);

double xmin{-2}, xmax{2};

double miny, maxy;

WCHAR mainWindowName[] = \_T("MainWindow");

int WINAPI WinMain(HINSTANCE This,

HINSTANCE Prev,

LPSTR cmd,

int mode) {

WNDCLASS wc;

memset(&wc, 0, sizeof(wc));

wc.hInstance = This;

wc.lpszClassName = mainWindowName;

wc.lpfnWndProc = MainWndProc;

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

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

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

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

RegisterClass(&wc);

HMENU menu = LoadMenu(NULL, (LPCWSTR)IDR\_MENU1);

auto static mainWindow = CreateWindow(mainWindowName,

\_T("SP lab 3"),

WS\_OVERLAPPEDWINDOW,

(GetSystemMetrics(SM\_CXSCREEN) - 500) / 2,

(GetSystemMetrics(SM\_CYSCREEN) - 500) / 2,

500,

500,

HWND\_DESKTOP,

menu,

This,

NULL);

ShowWindow(mainWindow, mode);

MSG msg;

while (GetMessage(&msg, mainWindow, 0, 0)) {

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT msg,

WPARAM wParam, LPARAM lParam) {

typedef void (drawFunc)(HWND hwnd, HDC hdc, double exmin, double exmax);

typedef xpair(showFunc)(HWND hwnd, int exmin, int exmax);

static drawFunc \*draw\_frame = NULL;

static showFunc \*show\_dialog = NULL;

static HINSTANCE hmodule = NULL;

static HINSTANCE smodule = NULL;

switch (msg) {

case WM\_CREATE: {

hmodule = LoadLibrary(\_T("E:/Projects/VisualStudioProjects/Graphics\_lab/x64/Debug/DrawGraphic.dll"));

if (hmodule == NULL) {

SetWindowText(hWnd, \_T("Module not loaded"));

MessageBox(NULL,

(LPCWSTR)L"Graphic DLL isn't loaded!",

(LPCWSTR)L"DLL error",

MB\_ICONWARNING);

ExitProcess(-1);

}

else {

std::cout << hmodule << std::endl;

draw\_frame = (drawFunc\*)GetProcAddress(hmodule, "drawGraphic");

if (draw\_frame != NULL) {

SetWindowText(hWnd, \_T("Dll func loaded"));

}

else {

SetWindowText(hWnd, \_T("Error load func from dll"));

MessageBox(NULL,

(LPCWSTR)L"Graphic function from DLL isn't loaded!",

(LPCWSTR)L"DLL func error",

MB\_ICONWARNING);

ExitProcess(-1);

}

}

}

case WM\_PAINT: {

PaintGuard guard{ hWnd };

HDC hdc = guard.GetHDC();

WindowSize ws = GetWindowSize(hWnd);

draw\_frame(hWnd, hdc, xmin, xmax);

//DrawFunc(hdc, ws);

//int zeroy = DrawAxisX(hdc, ws);

//DrawAxisY(hdc, ws, zeroy);

break;

}

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case ID\_DW:

smodule = LoadLibrary(\_T("E:/Projects/VisualStudioProjects/Graphics\_lab/x64/Debug/ShowDialogDll.dll"));

if (smodule == NULL) {

SetWindowText(hWnd, \_T("Module not loaded"));

MessageBox(NULL,

(LPCWSTR)L"Dialog DLL isn't loaded!",

(LPCWSTR)L"DLL error",

MB\_ICONWARNING);

HMENU hmenu = GetMenu(hWnd);

EnableMenuItem(hmenu, ID\_DW, MF\_DISABLED);

SetMenu(hWnd, hmenu);

}

else {

show\_dialog = (showFunc\*)GetProcAddress(smodule, "showDialog");

//test\* show\_dialog = (test\*)GetProcAddress(smodule, "ret1");

if (show\_dialog != NULL) {

SetWindowText(hWnd, \_T("Dll func loaded"));

auto xarr = show\_dialog(hWnd, xmin, xmax);

xmin = xarr.xmin;

xmax = xarr.xmax;

//int x = show\_dialog();

}

else {

SetWindowText(hWnd, \_T("Error load func from dll"));

MessageBox(NULL,

(LPCWSTR)L"Dialog function from DLL isn't loaded!",

(LPCWSTR)L"DLL func error",

MB\_ICONWARNING);

HMENU hmenu = GetMenu(hWnd);

EnableMenuItem(hmenu, ID\_DW, MF\_DISABLED);

SetMenu(hWnd, hmenu);

}

}

//if (DialogBox(NULL, (LPCTSTR)IDD\_DIALOG1, hWnd,

// (DLGPROC)ParFun) == IDOK) {

// SetWindowText(hWnd, (\_T("min = ") + ToString(xmin) + \_T("; max = ") + ToString(xmax)).data());

// InvalidateRect(hWnd, NULL, TRUE);

//}

}

break;

case WM\_RBUTTONUP:

break;

case WM\_LBUTTONUP:

if (DialogBox(NULL, (LPCTSTR)IDD\_DIALOG1, hWnd,

(DLGPROC)ParFun) == IDOK) {

SetWindowText(hWnd, (\_T("min = ") + ToString(xmin) + \_T("; max = ") + ToString(xmax)).data());

InvalidateRect(hWnd, NULL, TRUE);

}

break;

case WM\_DESTROY:

FreeLibrary(hmodule);

FreeLibrary(smodule);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

LRESULT CALLBACK ParFun(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {

auto GetInt = [=](const int idc) {

String buffer(100, 0);

GetDlgItemText(hDlg, idc, &buffer[0], buffer.size());

return std::stoi(buffer);

};

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK) {

xmin = GetInt(IDC\_EMin);

xmax = GetInt(IDC\_EMax);

EndDialog(hDlg, IDOK);

return TRUE;

}

if (LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, IDCANCEL);

return TRUE;

}

}

return 0;

}

//int DrawAxisX(HDC hdc, WindowSize ws) {

// int zeroy;

// int textPadding;

// TEXTMETRIC tm;

// GetTextMetrics(hdc, &tm);

// int letterHeight = tm.tmHeight;

// int letterWidth = tm.tmAveCharWidth;

// if (maxy <= 0) {

// textPadding = 10;

// zeroy = 7;

// MoveToEx(hdc, 0, zeroy, nullptr);

// LineTo(hdc, ws.width, zeroy);

// }

// else if (miny >= 0) {

// textPadding = -1 \* letterHeight;

// zeroy = ws.height - 7;

// MoveToEx(hdc, 0, zeroy, nullptr);

// LineTo(hdc, ws.width, zeroy);

// }

// else {

// textPadding = 10;

// zeroy = -miny \* ws.height / (maxy - miny);

// MoveToEx(hdc, 0, ws.height - zeroy, nullptr);

// LineTo(hdc, ws.width, ws.height - zeroy);

// zeroy = ws.height - zeroy;

// }

//

// double currentX = xmin;

// int linePadding = 7;

// double dx = (xmax - xmin) / ws.width;

// int step = letterWidth \* 12 + linePadding;

// for (int i = 0; i < ws.width; i += step) {

// std::wstring sx = std::to\_wstring(currentX);

// TextOut(hdc, i , zeroy + textPadding, sx.c\_str(), sx.size());

// MoveToEx(hdc, i, zeroy - linePadding, nullptr);

// LineTo(hdc, i, zeroy + linePadding);

// currentX += step \* dx;

// }

// return zeroy;

//}

//

//

//void DrawAxisY(HDC hdc, WindowSize ws, int zeroy) {

// int zerox;

// int textPadding;

// TEXTMETRIC tm;

// GetTextMetrics(hdc, &tm);

// int letterHeight = tm.tmHeight;

// int letterWidth = tm.tmAveCharWidth;

// int voidPadding = 2 \* letterHeight;

// if (xmax <= 0) {

// textPadding = -10 \* letterWidth;

// zerox = ws.width - 7;

// MoveToEx(hdc, zerox, 0, nullptr);

// LineTo(hdc, zerox, ws.height);

// } else if (xmin >= 0) {

// textPadding = 10;

// zerox = 7;

// MoveToEx(hdc, zerox, 0, nullptr);

// LineTo(hdc, zerox, ws.height);

// } else {

// textPadding = 10;

// zerox = -xmin \* ws.width / (xmax - xmin);

// MoveToEx(hdc, zerox, 0, nullptr);

// LineTo(hdc, zerox, ws.height);

// }

// if (miny >= 0) {

// zeroy -= voidPadding;

// }

// double currentY = miny;

// int linePadding = 7;

// double dy = (maxy - miny) / static\_cast<double>(ws.height);

// int step = letterHeight + linePadding;

// for (int i = ws.height; i >= 0; i -= step) {

// std::wstring sy = std::to\_wstring(currentY);

// if (!(zeroy < i && i < zeroy + voidPadding)) {

// TextOut(hdc, zerox + textPadding, i - letterHeight/2 , sy.c\_str(), sy.size());

// }

// MoveToEx(hdc, zerox - linePadding, i, nullptr);

// LineTo(hdc, zerox + linePadding, i);

// currentY += step \* dy;

// }

//}

//

//

//std::vector<int> TransformValues(std::vector<double> oldValues, WindowSize ws) {

// std::vector<int> result{};

// for (double oldy : oldValues) {

// int newy = TransformValue(oldy, ws.height, miny, maxy);

// result.push\_back(newy);

// }

// return result;

//}

//

//

//int TransformValue(double x, int newSize, double min, double max) {

// int result = newSize - (x - min) / (max - min) \* newSize;

// return result;

//}

//

//

//

//void DrawFunc(HDC hdc, WindowSize ws) {

// auto funcValues = CalculateFuncValues(xmin, xmax, ws.width);

// miny = \*std::min\_element(funcValues.begin(), funcValues.end());

// maxy = \*std::max\_element(funcValues.begin(), funcValues.end());

// auto values = TransformValues(funcValues, ws);

// MoveToEx(hdc, 0, values[0], nullptr);

// for (int i = 1; i < values.size(); ++i) {

// LineTo(hdc, i, values[i]);

// }

//}

//

//

//double CalculateFunc(double x) {

// return sin(x) + cos(x);

//}

//

//

//std::vector<double> CalculateFuncValues(double xmin, double xmax, int steps) {

// std::vector<double> result{};

// double x = xmin;

// for (int i = 0; i < steps; ++i, x += (xmax - xmin) / steps) {

// result.push\_back(CalculateFunc(x));

// }

// return result;

//}

**Resource1.h**

//{{NO\_DEPENDENCIES}}

// Включаемый файл, созданный в Microsoft Visual C++.

// Используется Resource.rc

//

#define IDD\_DIALOG1 101

#define IDR\_MENU1 103

#define IDC\_EMin 1005

#define IDC\_EMax 1006

#define ID\_40001 40001

#define ID\_DW 40002

// Next default values for new objects

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NEXT\_RESOURCE\_VALUE 104

#define \_APS\_NEXT\_COMMAND\_VALUE 40003

#define \_APS\_NEXT\_CONTROL\_VALUE 1008

#define \_APS\_NEXT\_SYMED\_VALUE 101

#endif

#endif

**ShowDialogDll.cpp**

#include "ShowDialogDll.h"

#include "pch.h"

#include "utility.hpp"

#include "resource1.h"

#include <array>

static int xmin;

static int xmax;

LRESULT CALLBACK ParFun(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {

auto GetInt = [=](const int idc) {

String buffer(100, 0);

GetDlgItemText(hDlg, idc, &buffer[0], buffer.size());

return std::stoi(buffer);

};

switch (message)

{

case WM\_INITDIALOG:

return TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK) {

xmin = GetInt(IDC\_EMin);

xmax = GetInt(IDC\_EMax);

EndDialog(hDlg, IDOK);

return TRUE;

}

if (LOWORD(wParam) == IDCANCEL) {

EndDialog(hDlg, IDCANCEL);

return TRUE;

}

}

return 0;

}

struct xpair {

int xmin;

int xmax;

};

extern "C" \_declspec(dllexport) xpair showDialog(HWND hWnd, int xmin, int xmax) {

if (DialogBox(NULL, (LPCTSTR)IDD\_DIALOG1, hWnd,

(DLGPROC)ParFun) == IDOK) {

SetWindowText(hWnd, (\_T("min = ") + ToString(xmin) + \_T("; max = ") + ToString(xmax)).data());

InvalidateRect(hWnd, NULL, TRUE);

}

xpair xx;

xx.xmin = ::xmin;

xx.xmax = ::xmax;

return xx;

}

**Скриншоты**









