МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Институт КНТ

Факультет ФИСП

Кафедра ПИ им. Л.П. Фельдмана

Лабораторная работа №3

Тема: Интерфейс графических устройств

Курс: Системное программирование

Выполнил

ст. гр. ПИ-18б

Моргунов А.Г.

Проверил

Филипишин Д.А.

Донецк – 2021

**Задание**

Задание. Нарисовать график функции. Формула для построения графика выбирается в соответствии с вариантом. Пределы рисования графика определяются значениями xmin и xmax. Начальные значения этих параметров задаются при создании окна. Значения ymin и ymax должны соответствовать минимальному и максимальному значению функции на промежутке [xmin; xmax]. При построении функции левая граница клиентской области окна должна соответствовать xmin, правая – xmax, верхняя – ymax, нижняя – ymin. Нарисовать координатные оси, нанести на них деления и подписать. Количество делений подбирается таким образом, чтобы соответствовать размеру окна: подписи под осями ни при каких условиях не должны наезжать друг на друга. Координатные оси рисуются таким образом, чтобы их точка пересечения соответствовала x=0 и y=0. Если нуль находится вне экранной области, то координатная ось рисуется у соответствующей границы окна (например, если xmin<0 и xmax<0, то ось ординат рисуется у правой границы окна; если xmin>0 и xmax>0, то ось ординат рисуется у левой границы окна).

Добавить к программе пользовательский блок диалога. Поместить на него два текстовых поля (EditText) для ввода Xmin и Xmax (пределы рисования графика по X). Вызов блока диалога связать с событием «нажатие левой кнопки мыши в клиентской области окна» и выбором пункта меню, которое также необходимо добавить к программе.



**Код**

#define UNICODE

#include <iostream>

#include <vector>

#include <tchar.h>

#include <windows.h>

#include "resource1.h"

#include "utility.hpp"

#include "PaintGuard.hpp"

#include <algorithm>

#include <iterator>

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;

};

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) {

switch (msg) {

case WM\_PAINT: {

PaintGuard guard{ hWnd };

HDC hdc = guard.GetHDC();

WindowSize ws = GetWindowSize(hWnd);

DrawFunc(hdc, ws);

int zeroy = DrawAxisX(hdc, ws);

DrawAxisY(hdc, ws, zeroy);

break;

}

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case ID\_DW:

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:

CalculateFuncValues(-1, 1, 10);

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:

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.data(), 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 (xmin >= 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;

}

**Скриншоты**









