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 № 5

на тему

**РЕЕСТР И ЖУРНАЛЫ (WINDOWS)**

**ДОСТУП К РЕЕСТРУ WINDOWS. РАБОТА С ЖУРНАЛАМИ WINDOWS**

Выполнил:

студент гр. 153503

Татаринов В.В.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 6](#_Toc146836469)

[Выводы 7](#_Toc146836470)

[Список использованных источников 8](#_Toc146836471)

[Приложение А (обязательное) листинг кода 9](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Освоить реестр и журналы (*Windows*)*,* доступ к реестру *Windows*. Научиться работать с журналами *Windows*. Создать оконное приложение, для визуализации и анализа данных с использованием графиков и диаграмм, позволяющее сохранять данные графиков и диаграмм в реестре *Windows*.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Реестр — это определяемая системой база данных, в которой приложения и системные компоненты хранят и извлекают данные конфигурации. Данные, хранящиеся в реестре, зависят от версии *Microsoft* *Windows*. Приложения используют *API* реестра для получения, изменения или удаления данных реестра.

Реестр содержит информацию о программном обеспечении, установленных драйверах, настройках операционной системы и многом другом. Чтобы открыть реестр, можно использовать инструмент Редактор реестра [1].

Журналы событий в *Windows* – это специальные файлы, которые записывают информацию о различных событиях и ошибках, происходящих на компьютере или в операционной системе. Ведение журнала событий обеспечивает стандартный централизованный способ записи важных программных и аппаратных событий приложениями (и операционной системой). Служба ведения журнала событий записывает события из различных источников и сохраняет их в одной коллекции, называемой журналом событий [2].

* *RegCreateKeyEx* в *Win32* *API* используется для создания или открытия ключа в реестре *Windows*. Если ключ уже существует в реестре, то функция открывает его.
* *RegQueryValueEx* служит для получения значения указанного параметра из заданного ключа в реестре *Windows*. Возвращает тип и данные указанного значения по имени, ассоциирующимся с открытым ключом реестра. Если функция успешно, возвращается значение *ERROR*\_*SUCCESS*.
* *RegCloseKey* является функцией в операционных системах семейства *Windows*, которая используется для закрытия открытого ключа реестра *Windows* (*Registry* *Key*). Ключ реестра представляет собой структуру данных, используемую для хранения настроек и конфигураций операционной системы, программ и компонентов. Когда приложение открывает ключ реестра для чтения или записи данных, оно должно закрыть ключ, когда закончило работу с ним. Функция *RegCloseKey* используется для освобождения ресурсов, связанных с открытым ключом реестра, после его использования, и осуществляет закрытие ключа. Дескриптор указанного ключа не должен использоваться после того как он был закрыт, потому что он больше не действителен [3].

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы было создано приложение для визуализации и анализа данных с использованием графиков и диаграмм. Программа позволяет сохранять данные графиков и диаграмм в реестре *Windows*. Интерфейс программы представляет собой поля для ввода, столбчатую диаграмму и график (рисунок 1).

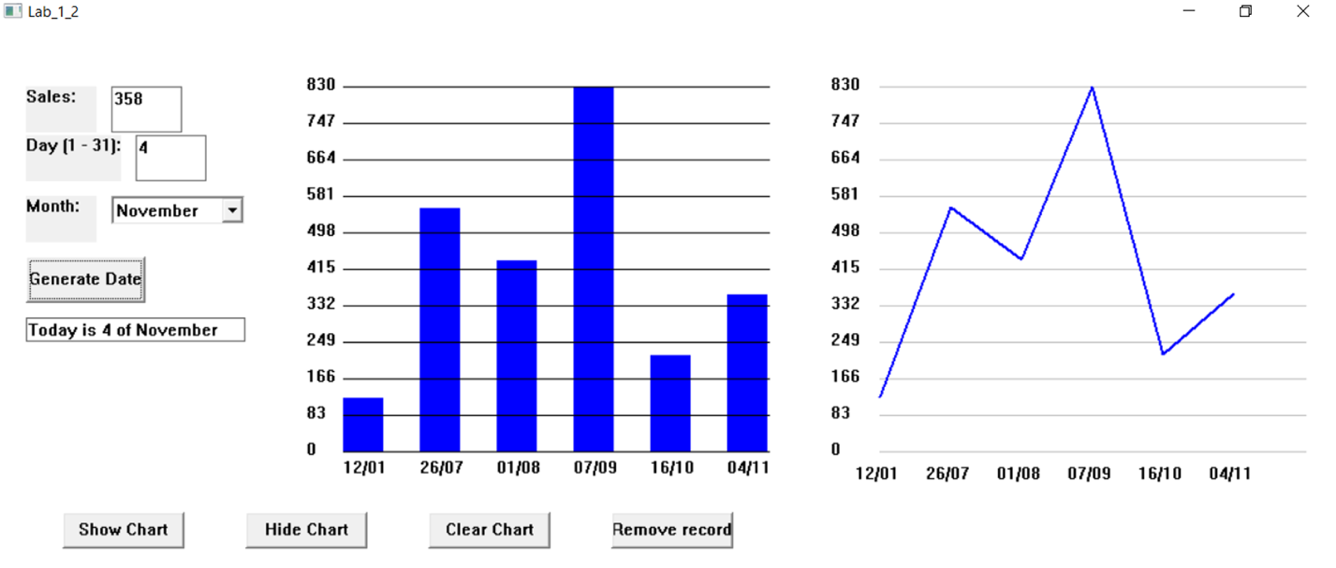


Рисунок 1 – Интерфейс приложения

Информация о продажах и датах автоматически сохраняется в реестр при закрытии приложении и автоматически подгружается при запуске приложения (рисунок 2).

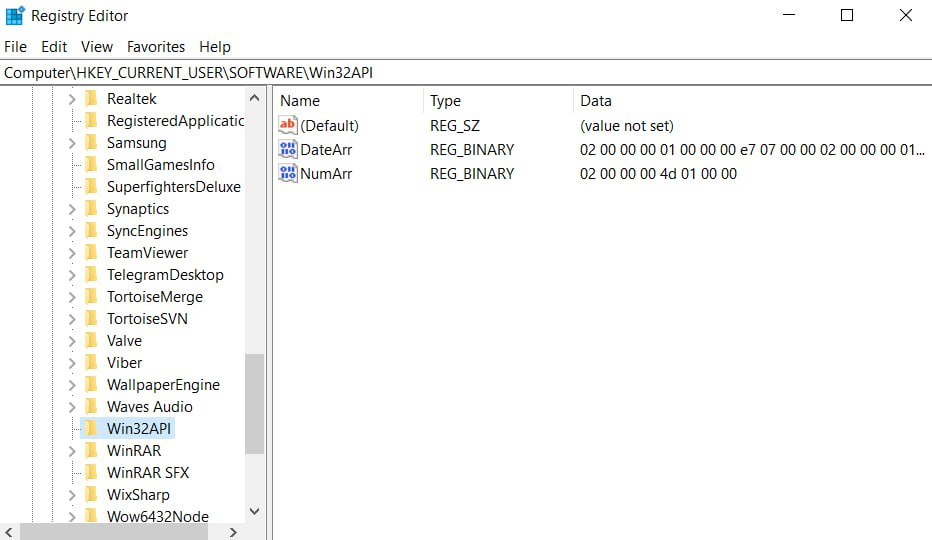


Рисунок 2 – Реестр

Информация о запуске приложения записывается при старте приложения в журнал (рисунок 3).

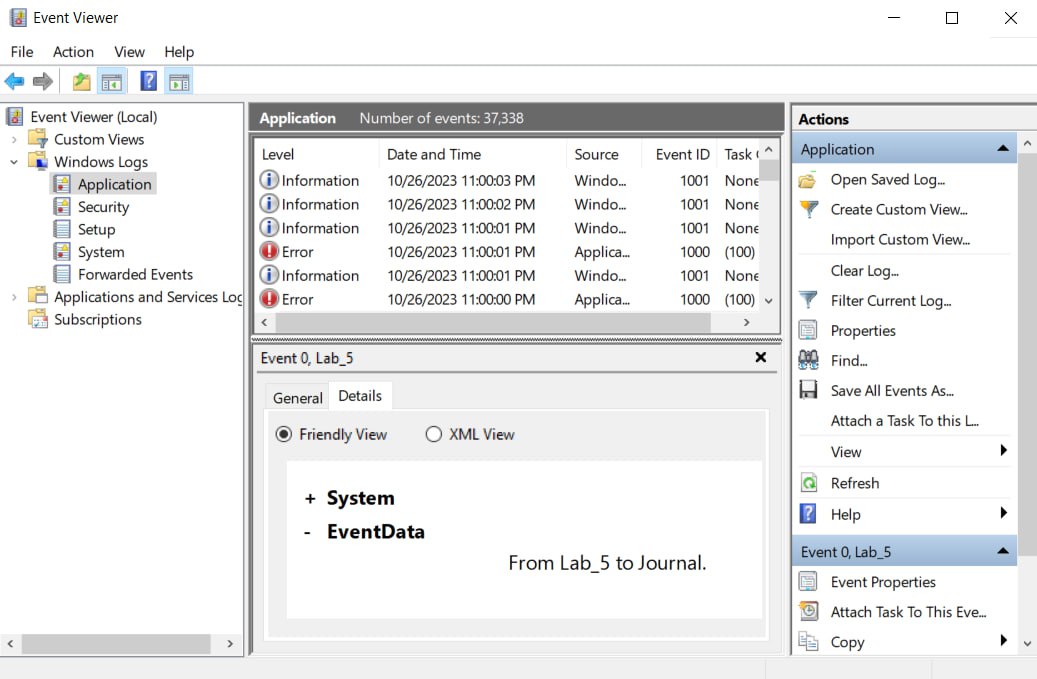


Рисунок 3 – Журнал событий

# **ВЫВОДЫ**

В результате выполнения лабораторной работы была освоена работа с реестром и журналами *Windows*. Создано оконное приложение, для визуализации и анализа данных с использованием графиков и диаграмм, позволяющее сохранять данные графиков и диаграмм в реестре *Windows*.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Реестр [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry>.
2. Ведение журнала событий [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/eventlog/event-logging/.
3. Функции доступа к реестру [Электронный ресурс]. – Режим доступа: <https://w32api.narod.ru/functions01.html>.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *Lab\_5\_WINAPI32.cpp*:

#define ID\_SHOW\_CHART 101

#define ID\_HIDE\_CHART 102

#define ID\_CLEAR\_CHART 103

#define ID\_REMOVE\_RECORD\_CHART 104

#define ID\_CREATE\_RECORD 105

#include <windows.h>

#include <regex>

#include <cstdlib>

#include <vector>

HMENU hMenu;

HWND hDay, hMonthList, hDate, hSales;

HHOOK hKeyboardHook = NULL;

HWND hActiveWindow = NULL;

bool showChart = false;

struct Date {

int day;

int month;

int year;

};

std::vector<Date> DateArr;

std::vector<int> NumArr;

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

void AddMenus(HWND hWnd);

void AddControls(HWND hWnd);

bool dateMatches(const Date& date, int day, int month, int year);

void CreateRecord(HWND hWnd);

void RemoveRecord(HWND hWnd);

void ClearChart();

bool Validate(std::wregex pattern, wchar\_t string[], int size, const wchar\_t err\_msg[], HWND hWnd, HWND hCustom);

void FillMonths();

void CreateGraphs(HWND hWnd);

void BuildColDiagram(HDC hdc);

void BuildPlot(HDC hdc);

void SaveDataToRegistry(const std::vector<int>& numArr, const std::vector<Date>& dateArr);

void LoadDataFromRegistry(std::vector<int>& numArr, std::vector<Date>& dateArr);

void WriteToEventLog(const std::wstring& message);

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN) {

KBDLLHOOKSTRUCT\* pKbStruct = (KBDLLHOOKSTRUCT\*)lParam;

if (pKbStruct->scanCode == 0x1C) {

hActiveWindow = GetForegroundWindow();

HWND hButton = GetDlgItem(hActiveWindow, ID\_CREATE\_RECORD);

HWND hButton\_2 = GetDlgItem(hActiveWindow, ID\_SHOW\_CHART);

if (hButton != NULL) {

SendMessage(hButton, BM\_CLICK, 0, 0);

}

if (hButton\_2 != NULL) {

SendMessage(hButton\_2, BM\_CLICK, 0, 0);

}

}

}

}

return CallNextHookEx(hKeyboardHook, nCode, wParam, lParam);

}

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR arts, int ncmdshow) {

WNDCLASSW wc = { 0 };

wc.hbrBackground = CreateSolidBrush(RGB(255, 255, 255));

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

wc.hInstance = hInst;

wc.lpszClassName = L"Class";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc))

return -1;

CreateWindowW(L"Class", L"Lab\_1\_2", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 20, 20, 1800, 500, NULL, NULL, NULL, NULL);

HHOOK hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, hInst, 0);

std::wstring messageToWrite = L"From Lab\_5 to Journal.";

WriteToEventLog(messageToWrite);

MSG msg = { 0 };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hKeyboardHook);

return 0;

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg)

{

case WM\_COMMAND:

switch (wp)

{

case ID\_CREATE\_RECORD: {

CreateRecord(hWnd);

InvalidateRect(hWnd, NULL, TRUE);

break;

}

case ID\_SHOW\_CHART: {

showChart = true;

InvalidateRect(hWnd, NULL, TRUE);

break;

}

case ID\_HIDE\_CHART: {

showChart = false;

InvalidateRect(hWnd, NULL, TRUE);

break;

}

case ID\_CLEAR\_CHART: {

showChart = false;

ClearChart();

InvalidateRect(hWnd, NULL, TRUE);

break;

}

case ID\_REMOVE\_RECORD\_CHART: {

RemoveRecord(hWnd);

break;

}

}

break;

case WM\_PAINT:

CreateGraphs(hWnd);

break;

case WM\_CREATE:

AddMenus(hWnd);

AddControls(hWnd);

LoadDataFromRegistry(NumArr, DateArr);

break;

case WM\_DESTROY:

SaveDataToRegistry(NumArr, DateArr);

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

}

void AddMenus(HWND hWnd) {

hMenu = CreateMenu();

SetMenu(hWnd, hMenu);

}

void AddControls(HWND hWnd) {

CreateWindowW(L"Static", L"Sales:", WS\_VISIBLE | WS\_CHILD, 20, 50, 58, 38, hWnd, NULL, NULL, NULL);

hSales = CreateWindowW(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 90, 50, 58, 38, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Static", L"Day (1 - 31):", WS\_VISIBLE | WS\_CHILD, 20, 90, 78, 38, hWnd, NULL, NULL, NULL);

hDay = CreateWindowW(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 110, 90, 58, 38, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Static", L"Month:", WS\_VISIBLE | WS\_CHILD, 20, 140, 58, 38, hWnd, NULL, NULL, NULL);

hMonthList = CreateWindowW(L"ComboBox", NULL, WS\_VISIBLE | WS\_CHILD | CBS\_DROPDOWNLIST, 90, 140, 110, 250, hWnd, NULL, NULL, NULL);

FillMonths();

CreateWindowW(L"Button", L"Generate Date", WS\_VISIBLE | WS\_CHILD, 20, 190, 98, 38, hWnd, (HMENU)ID\_CREATE\_RECORD, NULL, NULL);

hDate = CreateWindowW(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 20, 240, 180, 20, hWnd, NULL, NULL, NULL);

CreateWindowW(L"Button", L"Show Chart", WS\_VISIBLE | WS\_CHILD, 50, 400, 100, 30, hWnd, (HMENU)ID\_SHOW\_CHART, NULL, NULL);

CreateWindowW(L"Button", L"Hide Chart", WS\_VISIBLE | WS\_CHILD, 200, 400, 100, 30, hWnd, (HMENU)ID\_HIDE\_CHART, NULL, NULL);

CreateWindowW(L"Button", L"Clear Chart", WS\_VISIBLE | WS\_CHILD, 350, 400, 100, 30, hWnd, (HMENU)ID\_CLEAR\_CHART, NULL, NULL);

}

bool DateMatches(const Date& date, int day, int month, int year) {

return (date.day == day && date.month == month && date.year == year);

}

void CreateRecord(HWND hWnd) {

wchar\_t day[100], month[12], date[30], sales[100];

std::wregex day\_pattern(L"^(0?[1-9]|[12][0-9]|3[01])$");

if (!Validate(day\_pattern, day, 100, L"Invalid day input. Please enter a number between 1 and 31.", hWnd, hDay))

return;

std::wregex sales\_pattern(L"^(1000|([0-9]|[1-9][0-9]{0,2}))$");

if (!Validate(sales\_pattern, sales, 100, L"Invalid sales input. Please enter a number between 0 and 1000.", hWnd, hSales))

return;

int dayAsInt = \_wtoi(day);

int selectedMonthIndex = SendMessageW(hMonthList, CB\_GETCURSEL, 0, 0);

int salesAsInt = \_wtoi(sales);

if (DateArr.size() < 9) {

DateArr.push\_back(Date{ dayAsInt, selectedMonthIndex + 1, 2023 });

NumArr.push\_back(salesAsInt);

}

SendMessageW(hMonthList, CB\_GETLBTEXT, selectedMonthIndex, (LPARAM)month);

if (wcslen(day) > 0 && wcslen(month) > 0) {

wcscpy\_s(date, L"Today is ");

wcscat\_s(date, day);

wcscat\_s(date, L" of ");

wcscat\_s(date, month);

}

else {

wcscpy\_s(date, L"Empty field(s) exist");

}

SetWindowText(hDate, date);

}

void RemoveRecord(HWND hWnd) {

if (DateArr.size() < 0) {

return;

}

wchar\_t day[100], month[12], date[30];

std::wregex day\_pattern(L"^(0?[1-9]|[12][0-9]|3[01])$");

if (!Validate(day\_pattern, day, 100, L"Invalid day input. Please enter a number between 1 and 31.", hWnd, hDay))

return;

int dayAsInt = \_wtoi(day);

int selectedMonthIndex = SendMessageW(hMonthList, CB\_GETCURSEL, 0, 0);

auto it = std::find\_if(DateArr.begin(), DateArr.end(),

[&](const Date& date) {

return DateMatches(date, dayAsInt, selectedMonthIndex + 1, 2023);

});

if (it != DateArr.end()) {

int index = std::distance(DateArr.begin(), it);

DateArr.erase(DateArr.begin() + index);

NumArr.erase(NumArr.begin() + index);

}

InvalidateRect(hWnd, NULL, TRUE);

}

void CreateGraphs(HWND hWnd) {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

if (showChart == false || DateArr.size() == 0)

return;

BuildColDiagram(hdc);

BuildPlot(hdc);

EndPaint(hWnd, &ps);

}

void BuildColDiagram(HDC hdc) {

int MarginLeft = 280;

int MarginTop = 50;

int Width = 350;

int Height = 300;

int TotalWidth = Width + MarginLeft;

int TotalHeight = Height + MarginTop;

int ColumnSpacing = 30;

int BarWidth = (Width - (DateArr.size() - 1) \* ColumnSpacing) / DateArr.size();

int MaxValue = \*std::max\_element(NumArr.begin(), NumArr.end());

for (int i = 0; i < DateArr.size(); ++i) {

int barHeight = (NumArr[i] \* Height) / MaxValue;

int x = MarginLeft + (BarWidth + ColumnSpacing) \* i;

int y = MarginTop + Height - barHeight;

int barColor = RGB(0, 0, 255);

RECT rect = { x, y, x + BarWidth, MarginTop + Height };

HBRUSH hBrush = CreateSolidBrush(barColor);

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

WCHAR dateText[64];

swprintf\_s(dateText, \_countof(dateText), L"%02d/%02d", DateArr[i].day, DateArr[i].month);

TextOut(hdc, x, MarginTop + Height + 5, dateText, lstrlen(dateText));

}

for (int i = 0; i <= 10; ++i) {

int yLabel = MarginTop + Height - (i \* Height / 10);

WCHAR label[16];

swprintf\_s(label, \_countof(label), L"%d", i \* MaxValue / 10);

TextOut(hdc, MarginLeft - 30, yLabel - 10, label, lstrlen(label));

MoveToEx(hdc, MarginLeft, yLabel, nullptr);

LineTo(hdc, MarginLeft + Width, yLabel);

}

}

void BuildPlot(HDC hdc) {

int MarginLeft = 720;

int MarginTop = 50;

int Width = 350;

int Height = 300;

int TotalWidth = Width + MarginLeft;

int TotalHeight = Height + MarginTop;

int ColumnSpacing = 30;

int MaxValue = \*std::max\_element(NumArr.begin(), NumArr.end());

float stepX = 0;

if (DateArr.size() == 1) {

stepX = Width;

}

else {

stepX = static\_cast<float>(Width / DateArr.size());

}

float scaleY = static\_cast<float>(Height) / MaxValue;

int graphX = MarginLeft;

int graphY = MarginTop;

HPEN grayPen = CreatePen(PS\_SOLID, 1, RGB(192, 192, 192));

SelectObject(hdc, grayPen);

for (int i = 0; i <= 10; ++i) {

int yGridLine = graphY + Height - (i \* Height / 10);

MoveToEx(hdc, graphX, yGridLine, nullptr);

LineTo(hdc, graphX + Width, yGridLine);

WCHAR yAxisLabel[16];

swprintf\_s(yAxisLabel, \_countof(yAxisLabel), L"%d", i \* MaxValue / 10);

TextOut(hdc, graphX - 40, yGridLine - 10, yAxisLabel, lstrlen(yAxisLabel));

}

for (int i = 0; i < DateArr.size(); i++) {

int xLabel = static\_cast<int>(graphX + i \* stepX);

WCHAR label[16];

swprintf\_s(label, \_countof(label), L"%02d/%02d", DateArr[i].day, DateArr[i].month);

TextOut(hdc, xLabel - 20, graphY + Height + 10, label, lstrlen(label));

}

HPEN bluePen = CreatePen(PS\_SOLID, 2, RGB(0, 0, 255));

SelectObject(hdc, bluePen);

MoveToEx(hdc, graphX, graphY + Height - NumArr[0] \* scaleY, NULL);

for (int i = 0; i < DateArr.size(); i++) {

int x = static\_cast<int>(graphX + (i)\*stepX);

int y = graphY + Height - NumArr[i] \* scaleY;

LineTo(hdc, x, y);

}

DeleteObject(bluePen);

DeleteObject(grayPen);

}

void ClearChart() {

DateArr.clear();

NumArr.clear();

}

bool Validate(std::wregex pattern, wchar\_t string[], int size, const wchar\_t err\_msg[], HWND hWnd, HWND hCustom) {

GetWindowText(hCustom, string, size);

if (!std::regex\_match(string, pattern)) {

MessageBox(hWnd, err\_msg, L"Error", MB\_ICONERROR);

SetWindowText(hCustom, L"");

return false;

}

return true;

}

void FillMonths() {

const wchar\_t\* months[] = { L"January", L"February", L"March", L"April", L"May", L"June", L"July", L"August", L"September", L"October", L"November", L"December" };

for (int i = 0; i < 12; i++) {

SendMessageW(hMonthList, CB\_ADDSTRING, 0, (LPARAM)months[i]);

}

SendMessageW(hMonthList, CB\_SETCURSEL, 0, 0);

}

void SaveDataToRegistry(const std::vector<int>& numArr, const std::vector<Date>& dateArr) {

HKEY hKey;

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, L"Software\\Win32API", 0, NULL, REG\_OPTION\_NON\_VOLATILE, KEY\_WRITE, NULL, &hKey, NULL) == ERROR\_SUCCESS) {

RegSetValueEx(hKey, L"NumArr", 0, REG\_BINARY, reinterpret\_cast<const BYTE\*>(numArr.data()), numArr.size() \* sizeof(int));

RegSetValueEx(hKey, L"DateArr", 0, REG\_BINARY, reinterpret\_cast<const BYTE\*>(dateArr.data()), dateArr.size() \* sizeof(Date));

RegCloseKey(hKey);

}

}

void LoadDataFromRegistry(std::vector<int>& numArr, std::vector<Date>& dateArr) {

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software\\Win32API", 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

DWORD dataSize = 0;

if (RegQueryValueEx(hKey, L"NumArr", 0, NULL, NULL, &dataSize) == ERROR\_SUCCESS) {

numArr.resize(dataSize / sizeof(int));

RegQueryValueEx(hKey, L"NumArr", 0, NULL, reinterpret\_cast<BYTE\*>(numArr.data()), &dataSize);

}

dataSize = 0;

if (RegQueryValueEx(hKey, L"DateArr", 0, NULL, NULL, &dataSize) == ERROR\_SUCCESS) {

dateArr.resize(dataSize / sizeof(Date));

RegQueryValueEx(hKey, L"DateArr", 0, NULL, reinterpret\_cast<BYTE\*>(dateArr.data()), &dataSize);

}

RegCloseKey(hKey);

}

}

void WriteToEventLog(const std::wstring& message) {

HANDLE hEventLog = RegisterEventSource(NULL, L"Lab\_5");

if (hEventLog) {

const wchar\_t\* messageArray[1] = { message.c\_str() };

ReportEvent(hEventLog, EVENTLOG\_INFORMATION\_TYPE, 0, 0, NULL, 1, 0, messageArray, NULL);

DeregisterEventSource(hEventLog);

}

}