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

ОТЧЁТ

к лабораторной работе №5

на тему

РЕЕСТР И ЖУРНАЛЫ (WINDOWS). ДОСТУП К РЕЕСТРУ WINDOWS. РАБОТА С ЖУРНАЛАМИ WINDOWS. ДРУГИЕ ВСПОМОГАТЕЛЬНЫЕ СРЕДСТВА УПРАВЛЕНИЯ.

Выполнил студент гр.153502 Богомолов М.А.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи………………………………………………………….. 3](#_30j0zll)

[2 Теоретические сведения……………………………………………………….. 4](#_1fob9te)

[3 Описание функций программы………………………………………………... 5](#_3znysh7)

[Список использованных источников……………………………………………. 8](#_2et92p0)

[Приложение А (обязательное) Листинг кода………………………………….. 9](#_tyjcwt)

# **1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является создать утилиту для массовой установки и удаления программ с использованием информации из реестра *Windows*.

В качестве задачи необходимо реализовать удаление (деинсталляцию) приложений, их установку с помощью выбранного файла инсталлятора, а также загрузку.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Реестр *Windows* (англ. *Windows Registry*), или системный реестр – иерархически построенная база данных параметров и настроек в большинстве операционных систем *Microsoft Windows*.

Реестр содержит информацию и настройки для аппаратного обеспечения, программного обеспечения, профилей пользователей, предустановки. Большинство изменений в Панели управления, ассоциации файлов, системные политики, список установленного ПО фиксируются в реестре.

Реестр *Windows* был введён для упорядочения информации, хранившейся до этого во множестве *INI*-файлов, обеспечения единого механизма (*API*) записи-чтения настроек и избавления от проблемы коротких имён, отсутствия разграничения прав доступа и медленного доступа к *ini*-файлам, хранящимся на файловой системе *FAT16*, имевшей серьёзные проблемы быстродействия при поиске файлов в каталогах с большим их количеством. Со временем (окончательно ­– с появлением файловой системы *NTFS*) трудности, решавшиеся реестром, исчезли, но реестр остался из-за обратной совместимости и присутствует во всех версиях *Windows*, включая последнюю. Поскольку сейчас не существует реальных предпосылок для использования подобного механизма, *Microsoft* *Windows* – единственная (не считая *ReactOS* и *eComStation*) операционная система из используемых сегодня, в которой используется механизм реестра операционной системы [1].

Для работы с реестром в программе используются функции *RegOpenKeyEx*, *RegEnumKeyEx*, *RegQueryValueEx*.

Функция *RegOpenKeyEx* (*winreg.h*) открывает указанный раздел реестра [2].

Функция *RegEnumKeyEx* (*winreg*.*h*) перечисляет подразделы указанного открытого раздела реестра. Функция получает сведения об одном подразделе при каждом вызове [3].

Функция *RegQueryValueExA* (*winreg*.*h*) извлекает тип и данные для указанного имени значения, связанного с открытым разделом реестра [4].

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

При нажатии на кнопку *Load* загружается список установленных в системе приложений. Результат выполнения операции *Load* (Рисунок 1).

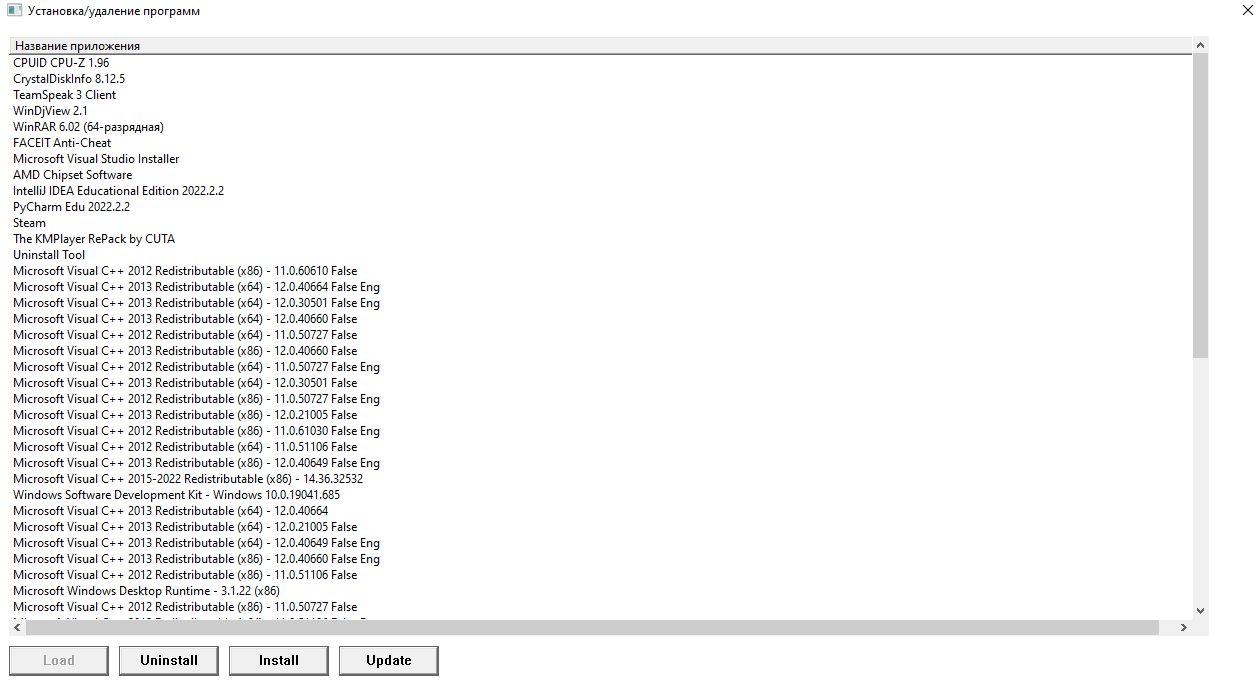


Рисунок 1 – Загрузка списка установленных приложений с помощью кнопки *Load*

При выборе конкретного элемента в поле *ListView* и нажатии кнопки *Uninstall* запускается деинсталлятор выбранного приложения. Результат выполнения операции (Рисунок 2).

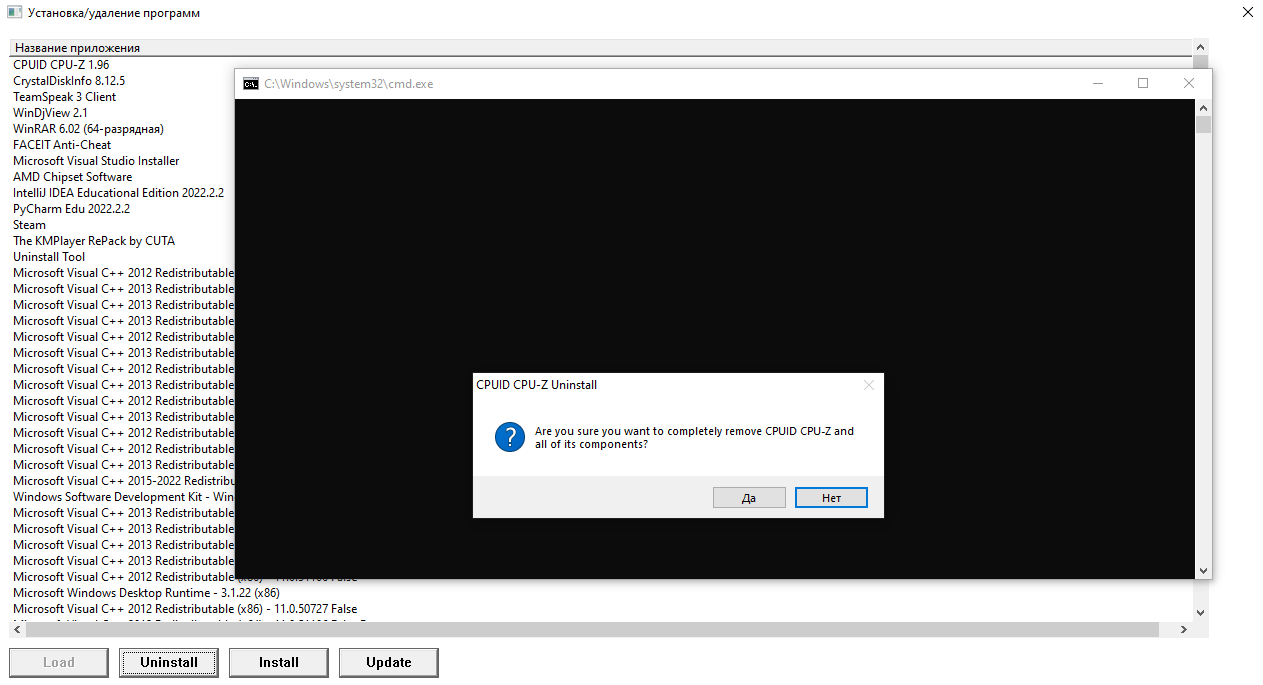
****

Рисунок 2 – Запуск деинсталлятора для приложения *CPUID CPU-Z* 1.96

При нажатии на кнопку *Update* обновляется список установленных в системе приложений.

При нажатии кнопки *Install* открывается диалоговое окно выбора файла, который должен являться установочным (Рисунок 3).

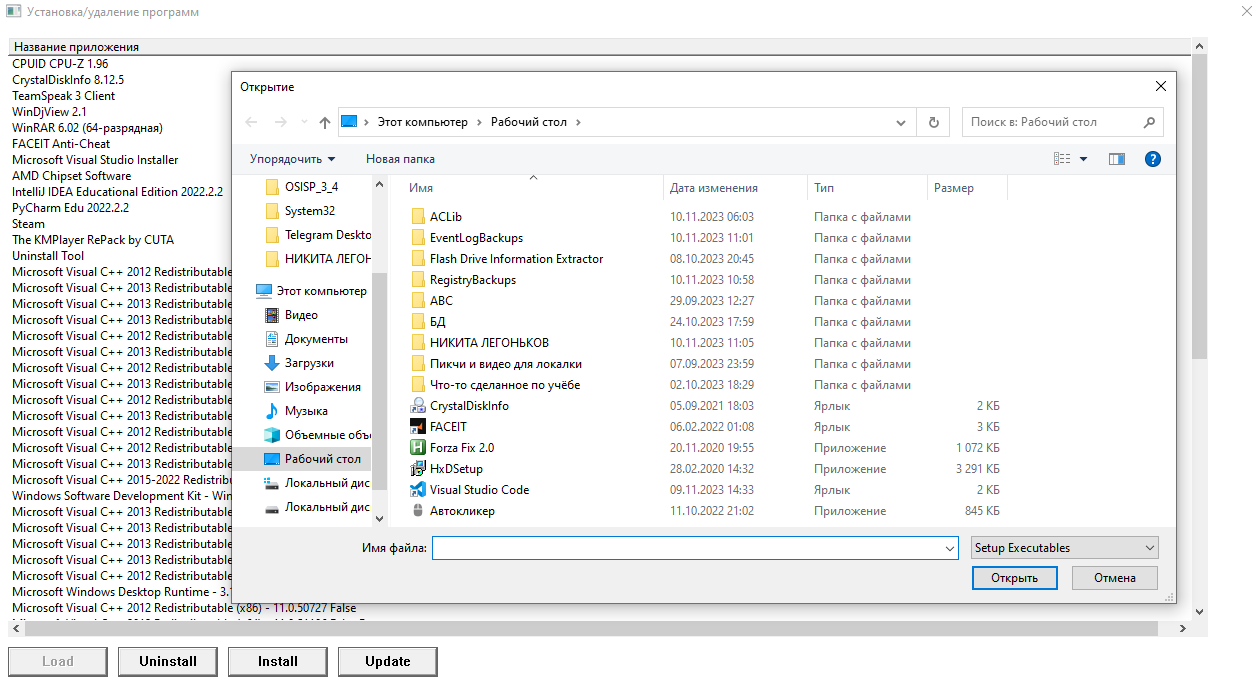
****

Рисунок 3 – Выбор установочного файла

При выборе конкретного файла в проводнике и нажатии кнопки Открыть происходит запуск установочного файла. Результат выбора установочного файла *HxDSetup*.*exe* (Рисунок 4).

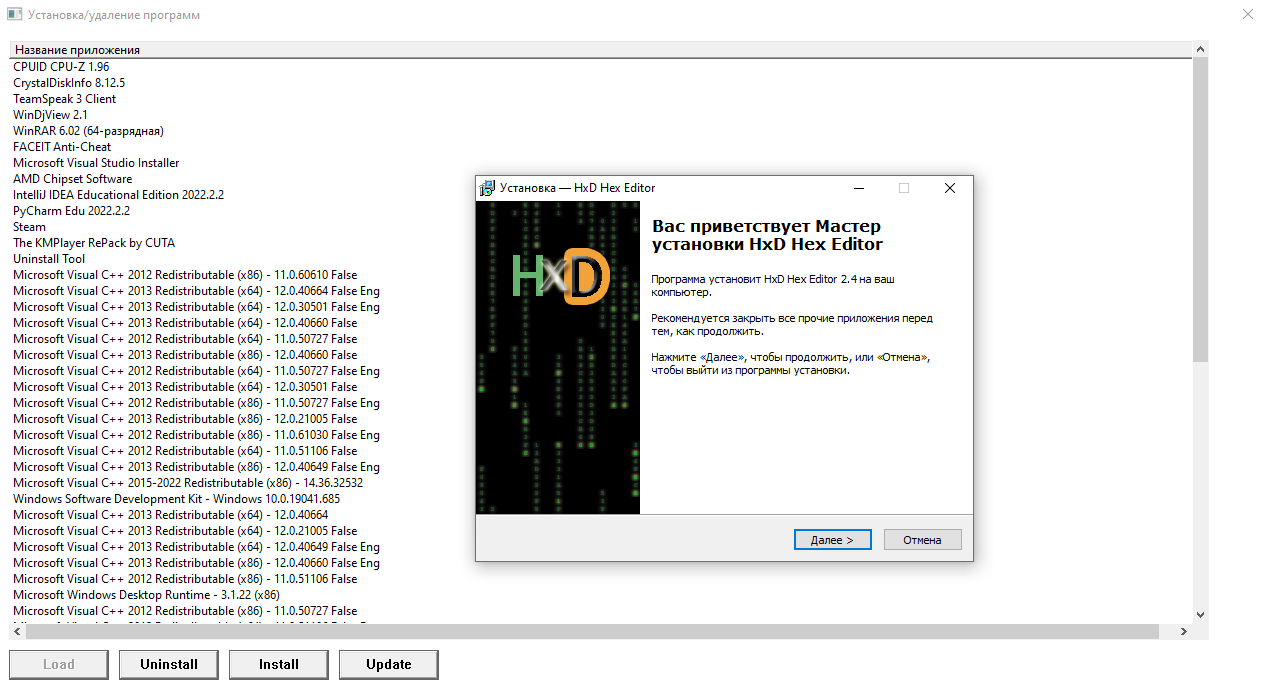
****

Рисунок 4 – Открытие установочного файла *HxDSetup.exe*

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Реестр Windows [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Реестр_Windows> – Дата доступа: 09.11.2023.

[2] Функция RegOpenKeyEx (winreg.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regopenkeyexa> – Дата доступа: 09.11.2023.

[3] Функция RegEnumKeyEx (winreg.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regenumkeyexa> – Дата доступа: 09.11.2023.

[4] Функция RegQueryValueEx (winreg.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/winreg/nf-winreg-regqueryvalueexa> – Дата доступа: 09.11.2023.

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

Листинг 1 – Файл OSISP\_5.cpp

#include <windows.h>

#include <CommCtrl.h>

#include <string>

#include <vector>

#include <thread>

#include <mutex>

#include <Shellapi.h>

#include <Msi.h>

#include <commdlg.h>

#include <shlobj.h>

#pragma comment(lib, "Comctl32.lib")

#pragma comment(lib, "Msi.lib")

#define IDC\_LISTVIEW 1000

#define IDC\_LOAD\_BUTTON 1001

#define IDC\_UNINSTALL\_BUTTON 1002

#define IDC\_INSTALL\_BUTTON 1003

#define IDC\_UPDATE\_BUTTON 1004

HWND listView;

HWND loadButton;

HWND uninstallButton;

HWND installButton;

HWND updateButton;

bool isMsgNeeded = false;

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

bool LaunchUninstallerForApp(DWORD registryBits, const std::wstring& appName);

void GetInstalledApps() {

std::vector<std::wstring> apps;

HKEY hKey64;

if (RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, KEY\_READ | KEY\_WOW64\_64KEY, &hKey64) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subKeyName[255];

DWORD subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

while (RegEnumKeyEx(hKey64, index, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY appKey;

if (RegOpenKeyEx(hKey64, subKeyName, 0, KEY\_READ, &appKey) == ERROR\_SUCCESS) {

WCHAR displayName[255];

DWORD displayNameSize = sizeof(displayName) / sizeof(displayName[0]);

if (RegQueryValueEx(appKey, L"DisplayName", NULL, NULL, (LPBYTE)displayName, &displayNameSize) == ERROR\_SUCCESS) {

WCHAR uninstallString[255];

DWORD uninstallStringSize = sizeof(uninstallString) / sizeof(uninstallString[0]);

if (RegQueryValueEx(appKey, L"UninstallString", NULL, NULL, (LPBYTE)uninstallString, &uninstallStringSize) == ERROR\_SUCCESS) {

if (\_wcsnicmp(uninstallString, L"MsiExec", 7) != 0) {

apps.push\_back(displayName);

}

}

}

RegCloseKey(appKey);

}

index++;

subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

}

RegCloseKey(hKey64);

}

HKEY hKey32;

if (RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, KEY\_READ | KEY\_WOW64\_32KEY, &hKey32) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subKeyName[255];

DWORD subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

while (RegEnumKeyEx(hKey32, index, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY appKey;

if (RegOpenKeyEx(hKey32, subKeyName, 0, KEY\_READ, &appKey) == ERROR\_SUCCESS) {

WCHAR displayName[255];

DWORD displayNameSize = sizeof(displayName) / sizeof(displayName[0]);

if (RegQueryValueEx(appKey, L"DisplayName", NULL, NULL, (LPBYTE)displayName, &displayNameSize) == ERROR\_SUCCESS) {

WCHAR uninstallString[255];

DWORD uninstallStringSize = sizeof(uninstallString) / sizeof(uninstallString[0]);

if (RegQueryValueEx(appKey, L"UninstallString", NULL, NULL, (LPBYTE)uninstallString, &uninstallStringSize) == ERROR\_SUCCESS) {

if (\_wcsnicmp(uninstallString, L"MsiExec", 7) != 0) {

apps.push\_back(displayName);

}

}

}

RegCloseKey(appKey);

}

index++;

subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

}

RegCloseKey(hKey32);

}

HKEY hKeyCurrentUser;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, KEY\_READ, &hKeyCurrentUser) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subKeyName[255];

DWORD subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

while (RegEnumKeyEx(hKeyCurrentUser, index, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY appKey;

if (RegOpenKeyEx(hKeyCurrentUser, subKeyName, 0, KEY\_READ, &appKey) == ERROR\_SUCCESS) {

WCHAR displayName[255];

DWORD displayNameSize = sizeof(displayName) / sizeof(displayName[0]);

if (RegQueryValueEx(appKey, L"DisplayName", NULL, NULL, (LPBYTE)displayName, &displayNameSize) == ERROR\_SUCCESS) {

WCHAR uninstallString[255];

DWORD uninstallStringSize = sizeof(uninstallString) / sizeof(uninstallString[0]);

if (RegQueryValueEx(appKey, L"UninstallString", NULL, NULL, (LPBYTE)uninstallString, &uninstallStringSize) == ERROR\_SUCCESS) {

if (\_wcsnicmp(uninstallString, L"MsiExec", 7) != 0) {

apps.push\_back(displayName);

}

}

}

RegCloseKey(appKey);

}

index++;

subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

}

RegCloseKey(hKeyCurrentUser);

}

ListView\_DeleteAllItems(listView);

for (size\_t i = 0; i < apps.size(); ++i) {

LVITEM lvItem;

lvItem.mask = LVIF\_TEXT;

lvItem.iItem = static\_cast<int>(i);

lvItem.iSubItem = 0;

lvItem.pszText = const\_cast<LPWSTR>(apps[i].c\_str());

ListView\_InsertItem(listView, &lvItem);

}

}

void UninstallSelectedApp() {

int selectedIndex = ListView\_GetNextItem(listView, -1, LVNI\_SELECTED);

if (selectedIndex != -1) {

LVITEM lvItem;

lvItem.iItem = selectedIndex;

lvItem.iSubItem = 0;

lvItem.mask = LVIF\_TEXT;

lvItem.cchTextMax = 255;

lvItem.pszText = new WCHAR[255];

ListView\_GetItem(listView, &lvItem);

std::wstring appName = lvItem.pszText;

delete[] lvItem.pszText;

if (LaunchUninstallerForApp(KEY\_WOW64\_64KEY, appName)) {

return;

}

if (LaunchUninstallerForApp(KEY\_WOW64\_32KEY, appName)) {

return;

}

}

}

bool LaunchUninstallerForApp(DWORD registryBits, const std::wstring& appName) {

HKEY hKey;

DWORD accessFlags = KEY\_READ | registryBits;

if (RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, accessFlags, &hKey) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subKeyName[255];

DWORD subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

while (RegEnumKeyEx(hKey, index, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY appKey;

if (RegOpenKeyEx(hKey, subKeyName, 0, KEY\_READ, &appKey) == ERROR\_SUCCESS) {

WCHAR displayName[255];

DWORD displayNameSize = sizeof(displayName) / sizeof(displayName[0]);

if (RegQueryValueEx(appKey, L"DisplayName", NULL, NULL, (LPBYTE)displayName, &displayNameSize) == ERROR\_SUCCESS) {

if (appName == displayName) {

WCHAR uninstallString[255];

DWORD uninstallStringSize = sizeof(uninstallString) / sizeof(uninstallString[0]);

if (RegQueryValueEx(appKey, L"UninstallString", NULL, NULL, (LPBYTE)uninstallString, &uninstallStringSize) == ERROR\_SUCCESS) {

HINSTANCE result = ShellExecute(0, L"open", uninstallString, 0, 0, SW\_SHOWNORMAL);

if ((intptr\_t)result > 32) {

return true;

}

\_wsystem(uninstallString);

RegCloseKey(appKey);

RegCloseKey(hKey);

return true;

}

}

}

RegCloseKey(appKey);

}

index++;

subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

}

RegCloseKey(hKey);

}

HKEY hKeyCurrentUser;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall", 0, accessFlags, &hKeyCurrentUser) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subKeyName[255];

DWORD subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

while (RegEnumKeyEx(hKeyCurrentUser, index, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY appKey;

if (RegOpenKeyEx(hKeyCurrentUser, subKeyName, 0, KEY\_READ, &appKey) == ERROR\_SUCCESS) {

WCHAR displayName[255];

DWORD displayNameSize = sizeof(displayName) / sizeof(displayName[0]);

if (RegQueryValueEx(appKey, L"DisplayName", NULL, NULL, (LPBYTE)displayName, &displayNameSize) == ERROR\_SUCCESS) {

if (appName == displayName) {

WCHAR uninstallString[255];

DWORD uninstallStringSize = sizeof(uninstallString) / sizeof(uninstallString[0]);

if (RegQueryValueEx(appKey, L"UninstallString", NULL, NULL, (LPBYTE)uninstallString, &uninstallStringSize) == ERROR\_SUCCESS) {

HINSTANCE result = ShellExecute(0, L"open", uninstallString, 0, 0, SW\_SHOWNORMAL);

if ((intptr\_t)result > 32) {

return true;

}

\_wsystem(uninstallString);

RegCloseKey(appKey);

RegCloseKey(hKeyCurrentUser);

return true;

}

}

}

RegCloseKey(appKey);

}

index++;

subKeyNameSize = sizeof(subKeyName) / sizeof(subKeyName[0]);

}

RegCloseKey(hKeyCurrentUser);

}

return false;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASS windowClass = {};

windowClass.lpfnWndProc = WindowProc;

windowClass.hInstance = GetModuleHandle(0);

windowClass.lpszClassName = L"WindowClass";

INITCOMMONCONTROLSEX icex;

icex.dwSize = sizeof(INITCOMMONCONTROLSEX);

icex.dwICC = ICC\_LISTVIEW\_CLASSES;

InitCommonControlsEx(&icex);

RegisterClass(&windowClass);

HWND hwnd = CreateWindowEx(

0,

L"WindowClass",

L"Установка/удаление программ",

WS\_OVERLAPPEDWINDOW & ~WS\_THICKFRAME & ~WS\_MAXIMIZEBOX & ~WS\_MINIMIZEBOX,

CW\_USEDEFAULT, CW\_USEDEFAULT,

1280, 700,

0, 0,

GetModuleHandle(0),

0

);

if (hwnd == NULL) {

return 1;

}

listView = CreateWindowEx(

0,

WC\_LISTVIEW,

L"",

WS\_CHILD | WS\_VISIBLE | LVS\_REPORT,

10, 10, 1200, 600,

hwnd,

(HMENU)IDC\_LISTVIEW,

GetModuleHandle(0),

0

);

LVCOLUMN lvColumn1;

lvColumn1.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

lvColumn1.fmt = LVCFMT\_LEFT;

lvColumn1.cx = 1200;

lvColumn1.pszText = const\_cast<LPWSTR>(L"Название приложения");

lvColumn1.iSubItem = 0;

ListView\_InsertColumn(listView, 0, &lvColumn1);

loadButton = CreateWindow(

L"BUTTON",

L"Load",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 620, 100, 30,

hwnd,

(HMENU)IDC\_LOAD\_BUTTON,

GetModuleHandle(0),

NULL

);

uninstallButton = CreateWindow(

L"BUTTON",

L"Uninstall",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

120, 620, 100, 30,

hwnd,

(HMENU)IDC\_UNINSTALL\_BUTTON,

GetModuleHandle(0),

NULL

);

installButton = CreateWindow(

L"BUTTON",

L"Install",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

230, 620, 100, 30,

hwnd,

(HMENU)IDC\_INSTALL\_BUTTON,

GetModuleHandle(0),

NULL

);

updateButton = CreateWindow(

L"BUTTON",

L"Update",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

340, 620, 100, 30,

hwnd,

(HMENU)IDC\_UPDATE\_BUTTON,

GetModuleHandle(0),

NULL

);

EnableWindow(installButton, FALSE);

EnableWindow(uninstallButton, FALSE);

EnableWindow(updateButton, FALSE);

ShowWindow(hwnd, SW\_SHOWNORMAL);

UpdateWindow(hwnd);

MSG msg = {};

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

void RunInstaller() {

COMDLG\_FILTERSPEC filters[] = { L"Setup Executables", L"\*.exe" };

IFileOpenDialog\* openFileDialog;

HRESULT result = CoCreateInstance(

CLSID\_FileOpenDialog,

NULL,

CLSCTX\_INPROC\_SERVER,

IID\_PPV\_ARGS(&openFileDialog)

);

if (SUCCEEDED(result)) {

openFileDialog->SetFileTypes(ARRAYSIZE(filters), filters);

result = openFileDialog->Show(NULL);

if (SUCCEEDED(result)) {

IShellItem\* selectedItem;

result = openFileDialog->GetResult(&selectedItem);

if (SUCCEEDED(result)) {

LPWSTR itemPath;

result = selectedItem->GetDisplayName(SIGDN\_FILESYSPATH, &itemPath);

if (SUCCEEDED(result)) {

ShellExecute(NULL, L"open", itemPath, NULL, NULL, SW\_SHOWNORMAL);

}

selectedItem->Release();

}

}

openFileDialog->Release();

}

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDC\_LOAD\_BUTTON:

std::thread(GetInstalledApps).detach();

EnableWindow(loadButton, FALSE);

EnableWindow(uninstallButton, TRUE);

EnableWindow(installButton, TRUE);

EnableWindow(updateButton, TRUE);

break;

case IDC\_UNINSTALL\_BUTTON:

std::thread(UninstallSelectedApp).detach();

isMsgNeeded = true;

break;

case IDC\_UPDATE\_BUTTON:

std::thread(GetInstalledApps).detach();

break;

case IDC\_INSTALL\_BUTTON:

MessageBox(nullptr, L"Для установки программы вам нужно выбрать установочный файл. Они обычно содержат в своём названии слово 'Setup'. ", L"Важное сообщение", MB\_OK | MB\_ICONINFORMATION);

RunInstaller();

isMsgNeeded = true;

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_ACTIVATE:

if (LOWORD(wParam) != WA\_INACTIVE && isMsgNeeded) {

MessageBox(nullptr, L"Для корректной работы программы вне зависимости от результата выполнения установки/деинсталляции требуется обновить список установленных программ с помощью кнопки Update.", L"Важное сообщение", MB\_OK | MB\_ICONINFORMATION);

isMsgNeeded = false;

}

break;

default:

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

}

return 0;

}