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. Другие вспомогательные средства управления.»

Выполнил:

студент гр. 153504

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты](#_1fob9te) 5

[Вывод](#_4q3p755gzys6) 8

[Список использованных источников](#_4q3p755gzys6) 9

[Приложение А (обязательное) Листинг кода](#_4q3p755gzys6) 10

**1 ЦЕЛИ РАБОТЫ**

1 Изучить реестр и журналы (Windows).

2 Разработать утилиту для создания и управления реестровыми записями Windows, включая создание, изменение и удаление ключей и значений.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Реестр Windows - это централизованное хранилище информации, используемое операционной системой Windows для хранения настроек, конфигураций, параметров и других данных, необходимых для работы системы и приложений. Реестр содержит иерархическую структуру, состоящую из ключей и их значениями.

Ключи ­ это контейнеры, которые могут содержать другие ключи и значения.

Ключи организованы в иерархическую структуру, с корневыми ключами на вершине.

Корневые ключи:

1 *HKEY\_CLASSES\_ROOT (HKCR).* Содержит информацию о типах файлов и зарегистрированных приложениях.

2 *HKEY\_CURRENT\_USER (HKCU)*. Содержит настройки, специфичные для текущего пользователя.

3 *HKEY\_LOCAL\_MACHINE (HKLM)*. Содержит настройки, общие для всех пользователей на компьютере.

4 *HKEY\_USERS (HKU).* Содержит профили пользователей, подключенные в данный момент или ранее.

5 *HKEY\_CURRENT\_CONFIG* (HKCC). Содержит информацию о текущей конфигурации оборудования.

Функции для работы с ключами реестра:

1 *RegCreateKeyEx*. Создает или открывает ключ в реестре.

2 *RegOpenKeyEx*. Открывает существующий ключ в реестре.

3 *RegCloseKey*. Закрывает открытый ключ.

4 *RegDeleteKey*. Удаляет указанный ключ и все его подключи и значения.

Ключи могут содержать значения, которые хранят фактическую информацию.

Значения могут быть разных типов, таких как строки, числа, бинарные данные и т.д.

Функции для работы со значениями ключей реестра:

1 *RegSetValueEx*. Устанавливает значение для указанного ключа.

2 *RegQueryValueEx*. Получает значение из указанного ключа.

3 *RegDeleteValue*. Удаляет указанное значение из ключа.

Важно помнить, что изменение реестра может повлиять на работу операционной системы и приложений, поэтому необходимо быть осторожным при внесении изменений.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

На рисунке 3.1 продемонстрирована функция добавления ключа в реестр. После нажатия на кнопку добавления в дереве реестра приложения и самом реестре появился новый ключ.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 3.1 – Добавление ключа в реестр

На рисунке 3.2 продемонстрирована удаления ключа из реестра. После нажатия на кнопку удаления будет удален введенный ключ в реестр, если такой есть.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 3.2 ­ Удаления ключа из реестра

На рисунке 3.3 продемонстрирована функция изменения имени ключа. При нахождении ключа с введенным именем оно будет заменено на имя введенное во втором поле ввода.

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 3.3 ­ Изменение имени ключа

На рисунке 3.4 продемонстрирована функция добавления значения ключа. При нахождении ключа с введенным именем будет добавлено имя значения и само значение в выбранный ключ.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 3.4 ­ Добавление(изменение) значения ключа

На рисунке 3.5 продемонстрирована функция удаления значения ключа. При нахождении ключа с введенным именем будет удалено значение с введенным именем.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 3.5 ­ Удаление значения ключа

.

**ВЫВОД**

В результате было разработано приложение для создания и управления реестровыми записями Windows, включая создание, изменение и удаление ключей и значений.

Для этого был изучен теоретический материал и средства Win32 Api, используемые для этих целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Реестр, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry>.

[2] Запись и удаление данных из реестра, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/writing-and-deleting-registry-data>.

[3] Открытие, создание и закрытие ключей, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/opening-creating-and-closing-keys>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <Windows.h>

#include <CommCtrl.h>

#include <string>

// Идентификаторы элементов управления

#define IDC\_TREEVIEW 1001

#define IDC\_LISTVIEW 1002

HWND hTreeView;

HWND hListView;

HWND g\_hEdit;

HWND g\_hEditNewName;

HWND g\_hAddButton;

HWND g\_hDeleteButton;

HWND g\_hRenameButton;

HWND g\_hEditKeyName;

HWND g\_hEditValueName;

HWND g\_hEditValueData;

HWND g\_hAddValueButton;

HWND g\_hDeleteValueButton;

HWND g\_hEditValueButton;

void TraverseRegistry(HWND hTreeView, HKEY hKey, HTREEITEM hParentItem);

void AddRegistryKey();

void DeleteRegistryKeyByName(HWND hEdit);

void RenameRegistryKey(HWND, HWND);

void AddRegistryValue(HWND, HWND, HWND);

void EditRegistryValue(HWND, HWND, HWND);

void DeleteRegistryValue(HWND, HWND);

// Обработчик сообщений главного окна

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

{

switch (message)

{

case WM\_COMMAND:

{

if (reinterpret\_cast<HWND>(lParam) == g\_hAddButton)

{

AddRegistryKey();

}

else if (reinterpret\_cast<HWND>(lParam) == g\_hDeleteButton)

{

DeleteRegistryKeyByName(g\_hEdit);

}

else if (reinterpret\_cast<HWND>(lParam) == g\_hRenameButton) {

RenameRegistryKey(g\_hEdit, g\_hEditNewName);

}

else if (reinterpret\_cast<HWND>(lParam) == g\_hAddValueButton) {

AddRegistryValue(g\_hEditKeyName, g\_hEditValueName, g\_hEditValueData);

}

else if (reinterpret\_cast<HWND>(lParam) == g\_hEditValueButton) {

EditRegistryValue(g\_hEditKeyName, g\_hEditValueName, g\_hEditValueData);

}

else if (reinterpret\_cast<HWND>(lParam) == g\_hDeleteValueButton) {

DeleteRegistryValue(g\_hEditKeyName, g\_hEditValueName);

}

break;

}

case WM\_CREATE:

{

// Создание элемента управления TreeView

hTreeView = CreateWindowEx(0, WC\_TREEVIEW, L"", WS\_CHILD | WS\_VISIBLE | TVS\_HASBUTTONS | TVS\_LINESATROOT | TVS\_EDITLABELS,

0, 0, 300, 400, hWnd, reinterpret\_cast<HMENU>(IDC\_TREEVIEW), nullptr, nullptr);

// Создание элемента управления ListView

hListView = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_LISTVIEW, L"", WS\_CHILD | WS\_VISIBLE | LVS\_REPORT,

320, 0, 300, 400, hWnd, reinterpret\_cast<HMENU>(IDC\_LISTVIEW), nullptr, nullptr);

// Добавление колонок в ListView

LVCOLUMN lvColumn{};

lvColumn.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvColumn.pszText = const\_cast<LPWSTR>(L"Name");

lvColumn.cx = 150;

ListView\_InsertColumn(hListView, 0, &lvColumn);

lvColumn.pszText = const\_cast <LPWSTR>(L"Value");

lvColumn.cx = 150;

ListView\_InsertColumn(hListView, 1, &lvColumn);

g\_hEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_EDIT, nullptr, WS\_VISIBLE | WS\_CHILD | ES\_AUTOHSCROLL, 620, 10, 150, 23, hWnd, nullptr, nullptr, nullptr);

g\_hEditNewName = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_EDIT, nullptr, WS\_VISIBLE | WS\_CHILD | ES\_AUTOHSCROLL, 620, 70, 150, 23, hWnd, nullptr, nullptr, nullptr);

g\_hEditKeyName = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_EDIT, nullptr, WS\_VISIBLE | WS\_CHILD | ES\_AUTOHSCROLL, 620, 100, 150, 23, hWnd, nullptr, nullptr, nullptr);

g\_hEditValueName = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_EDIT, nullptr, WS\_VISIBLE | WS\_CHILD | ES\_AUTOHSCROLL, 620, 130, 150, 23, hWnd, nullptr, nullptr, nullptr);

g\_hEditValueData = CreateWindowEx(WS\_EX\_CLIENTEDGE, WC\_EDIT, nullptr, WS\_VISIBLE | WS\_CHILD | ES\_AUTOHSCROLL, 620, 160, 150, 23, hWnd, nullptr, nullptr, nullptr);

g\_hAddButton = CreateWindowEx(0, WC\_BUTTON, L"Add", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 10, 75, 23, hWnd, nullptr, nullptr, nullptr);

g\_hDeleteButton = CreateWindowEx(0, WC\_BUTTON, L"Delete", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 40, 75, 23, hWnd, nullptr, nullptr, nullptr);

g\_hRenameButton = CreateWindowEx(0, WC\_BUTTON, L"Rename", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 70, 75, 23, hWnd, nullptr, nullptr, nullptr);

g\_hAddValueButton = CreateWindowEx(0, WC\_BUTTON, L"AddVal", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 100, 75, 23, hWnd, nullptr, nullptr, nullptr);

g\_hDeleteValueButton = CreateWindowEx(0, WC\_BUTTON, L"DelVal", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 130, 75, 23, hWnd, nullptr, nullptr, nullptr);

g\_hEditValueButton = CreateWindowEx(0, WC\_BUTTON, L"EditVal", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON, 780, 160, 75, 23, hWnd, nullptr, nullptr, nullptr);

HKEY hRootKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, nullptr, 0, KEY\_READ, &hRootKey) == ERROR\_SUCCESS)

{

TraverseRegistry(hTreeView, hRootKey, nullptr);

RegCloseKey(hRootKey);

}

break;

}

case WM\_NOTIFY:

{

NMHDR\* pHeader = reinterpret\_cast<NMHDR\*>(lParam);

if (pHeader->idFrom == IDC\_TREEVIEW)

{

if (pHeader->code == TVN\_SELCHANGED)

{

ListView\_DeleteAllItems(hListView);

HTREEITEM hSelectedItem = TreeView\_GetSelection(hTreeView);

if (hSelectedItem != nullptr)

{

TVITEM item{};

item.hItem = hSelectedItem;

item.mask = TVIF\_PARAM;

TreeView\_GetItem(hTreeView, &item);

item.mask = TVIF\_TEXT;

item.pszText = new wchar\_t[256];

item.cchTextMax = 256;

TreeView\_GetItem(hTreeView, &item);

std::wstring keyName = item.pszText;

HKEY hSelectedKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, keyName.c\_str(), 0, KEY\_READ | KEY\_QUERY\_VALUE, &hSelectedKey) == ERROR\_SUCCESS)

{

DWORD dwValueCount;

if (RegQueryInfoKey(hSelectedKey, nullptr, nullptr, nullptr, nullptr, nullptr, nullptr, &dwValueCount, nullptr, nullptr, nullptr, nullptr) == ERROR\_SUCCESS)

{

for (DWORD i = 0; i < dwValueCount; ++i)

{

wchar\_t valueName[256];

DWORD valueNameSize = sizeof(valueName) / sizeof(valueName[0]);

DWORD valueType;

BYTE data[1024];

DWORD dataSize = sizeof(data);

if (RegEnumValue(hSelectedKey, i, valueName, &valueNameSize, nullptr, &valueType, data, &dataSize) == ERROR\_SUCCESS)

{

LVITEM lvItem{};

lvItem.mask = LVIF\_TEXT;

lvItem.pszText = valueName;

lvItem.iItem = i;

std::wstring valueString;

switch (valueType)

{

case REG\_SZ:

case REG\_EXPAND\_SZ:

valueString = reinterpret\_cast<wchar\_t\*>(data);

break;

case REG\_DWORD:

valueString = std::to\_wstring(\*reinterpret\_cast<DWORD\*>(data));

break;

case REG\_QWORD:

valueString = std::to\_wstring(\*reinterpret\_cast<ULONGLONG\*>(data));

break;

default:

valueString = L"";

break;

}

lvItem.iSubItem = 0;

ListView\_InsertItem(hListView, &lvItem);

lvItem.mask = LVIF\_TEXT;

lvItem.pszText = const\_cast<LPWSTR>(valueString.c\_str());

lvItem.iItem = i; // Номер элемента

lvItem.iSubItem = 1; // Номер колонки

ListView\_SetItem(hListView, &lvItem);

int firstItemIndex = 0; // индекс первого элемента, который нужно перерисовать

int lastItemIndex = ListView\_GetItemCount(hListView) - 1; // индекс последнего элемента в списке

ListView\_RedrawItems(hListView, firstItemIndex, lastItemIndex);

}

}

}

else {

}

// Закрытие выбранного ключа

RegCloseKey(hSelectedKey);

}

delete[] item.pszText;

}

}

}

break;

}

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Функция для заполнения TreeView данными из реестра

void TraverseRegistry(HWND hTreeView, HKEY hKey, HTREEITEM hParentItem)

{

wchar\_t subKeyName[256];

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

// Перебор подключей в текущем ключе

for (DWORD i = 0; RegEnumKeyEx(hKey, i, subKeyName, &subKeyNameSize, nullptr, nullptr, nullptr, nullptr) == ERROR\_SUCCESS; ++i)

{

HKEY hSubKey;

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

{

TVINSERTSTRUCT tvInsert{};

tvInsert.hParent = hParentItem;

tvInsert.hInsertAfter = TVI\_LAST;

tvInsert.item.mask = TVIF\_TEXT | TVIF\_PARAM;

tvInsert.item.pszText = subKeyName;

tvInsert.item.lParam = reinterpret\_cast<LPARAM>(hSubKey);

HTREEITEM hNewItem = TreeView\_InsertItem(hTreeView, &tvInsert);

TraverseRegistry(hTreeView, hSubKey, hNewItem);

RegCloseKey(hSubKey);

}

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

}

}

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

{

const wchar\_t CLASS\_NAME[] = L"SampleWindowClass";

WNDCLASS wc{};

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hWnd = CreateWindowEx(0, CLASS\_NAME, L"Sample Window", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 900, 680, nullptr, nullptr, hInstance, nullptr);

if (hWnd == nullptr)

return 0;

ShowWindow(hWnd, nCmdShow);

MSG msg{};

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

void AddRegistryKey()

{

wchar\_t keyName[256];

GetWindowText(g\_hEdit, keyName, 256);

OutputDebugString(keyName);

HKEY hSoftwareKey;

RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_ALL\_ACCESS, &hSoftwareKey);

HKEY hNewKey;

LONG result = RegCreateKeyEx(hSoftwareKey, keyName, 0, nullptr, REG\_OPTION\_NON\_VOLATILE, KEY\_ALL\_ACCESS, nullptr, &hNewKey, nullptr);

RegCloseKey(hNewKey);

RegCloseKey(hSoftwareKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ключ успешно добавлен", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

TreeView\_DeleteAllItems(hTreeView);

HKEY hRootKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, nullptr, 0, KEY\_READ, &hRootKey) == ERROR\_SUCCESS)

{

TraverseRegistry(hTreeView, hRootKey, nullptr);

RegCloseKey(hRootKey);

}

}

void DeleteRegistryKeyByName(HWND hEdit)

{

int textLength = GetWindowTextLength(hEdit);

if (textLength == 0)

{

MessageBox(nullptr, L"Введите имя ключа", L"Ошибка", MB\_OK);

return;

}

std::wstring keyName(textLength + 1, L'\0');

GetWindowText(hEdit, &keyName[0], textLength + 1);

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_WRITE, &hParentKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия родительского ключа", L"Ошибка", MB\_OK);

return;

}

LONG result = RegDeleteKey(hParentKey, keyName.c\_str());

RegCloseKey(hParentKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ключ успешно удален", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else if (result == ERROR\_FILE\_NOT\_FOUND)

{

MessageBox(nullptr, L"Ключ с указанным именем не найден", L"Ошибка", MB\_OK);

}

else

{

MessageBox(nullptr, L"Ошибка при удалении ключа", L"Ошибка", MB\_OK);

}

TreeView\_DeleteAllItems(hTreeView);

HKEY hRootKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, nullptr, 0, KEY\_READ, &hRootKey) == ERROR\_SUCCESS)

{

TraverseRegistry(hTreeView, hRootKey, nullptr);

RegCloseKey(hRootKey);

}

}

void RenameRegistryKey(HWND hEditOldName, HWND hEditNewName)

{

int oldNameLength = GetWindowTextLength(hEditOldName);

if (oldNameLength == 0)

{

MessageBox(nullptr, L"Введите старое имя ключа", L"Ошибка", MB\_OK);

return;

}

int newNameLength = GetWindowTextLength(hEditNewName);

if (newNameLength == 0)

{

MessageBox(nullptr, L"Введите новое имя ключа", L"Ошибка", MB\_OK);

return;

}

std::wstring oldKeyName(oldNameLength + 1, L'\0');

std::wstring newKeyName(newNameLength + 1, L'\0');

GetWindowText(hEditOldName, &oldKeyName[0], oldNameLength + 1);

GetWindowText(hEditNewName, &newKeyName[0], newNameLength + 1);

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_WRITE, &hParentKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия родительского ключа", L"Ошибка", MB\_OK);

return;

}

LONG result = RegRenameKey(hParentKey, oldKeyName.c\_str(), newKeyName.c\_str());

RegCloseKey(hParentKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ключ успешно переименован", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else if (result == ERROR\_FILE\_NOT\_FOUND)

{

MessageBox(nullptr, L"Ключ с указанным старым именем не найден", L"Ошибка", MB\_OK);

}

else if (result == ERROR\_ALREADY\_EXISTS)

{

MessageBox(nullptr, L"Ключ с указанным новым именем уже существует", L"Ошибка", MB\_OK);

}

else

{

MessageBox(nullptr, L"Ошибка при переименовании ключа", L"Ошибка", MB\_OK);

}

TreeView\_DeleteAllItems(hTreeView);

HKEY hRootKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, nullptr, 0, KEY\_READ, &hRootKey) == ERROR\_SUCCESS)

{

TraverseRegistry(hTreeView, hRootKey, nullptr);

RegCloseKey(hRootKey);

}

}

void AddRegistryValue(HWND hEditKeyName, HWND hEditValueName, HWND hEditValueData)

{

int keyNameLength = GetWindowTextLength(hEditKeyName);

if (keyNameLength == 0)

{

MessageBox(nullptr, L"Введите имя ключа", L"Ошибка", MB\_OK);

return;

}

int valueNameLength = GetWindowTextLength(hEditValueName);

if (valueNameLength == 0)

{

MessageBox(nullptr, L"Введите имя данных", L"Ошибка", MB\_OK);

return;

}

int valueDataLength = GetWindowTextLength(hEditValueData);

if (valueDataLength == 0)

{

MessageBox(nullptr, L"Введите значение данных", L"Ошибка", MB\_OK);

return;

}

std::wstring keyName(keyNameLength + 1, L'\0');

std::wstring valueName(valueNameLength + 1, L'\0');

std::wstring valueData(valueDataLength + 1, L'\0');

GetWindowText(hEditKeyName, &keyName[0], keyNameLength + 1);

GetWindowText(hEditValueName, &valueName[0], valueNameLength + 1);

GetWindowText(hEditValueData, &valueData[0], valueDataLength + 1);

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_SET\_VALUE, &hParentKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия родительского ключа", L"Ошибка", MB\_OK);

return;

}

HKEY hKey;

if (RegOpenKeyEx(hParentKey, keyName.c\_str(), 0, KEY\_SET\_VALUE, &hKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия ключа", L"Ошибка", MB\_OK);

return;

}

LONG result = RegSetValueEx(hKey, valueName.c\_str(), 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(valueData.c\_str()), (valueDataLength + 1) \* sizeof(wchar\_t));

RegCloseKey(hKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Данные успешно добавлены или обновлены", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(nullptr, L"Ошибка при добавлении или обновлении данных", L"Ошибка", MB\_OK);

}

}

void EditRegistryValue(HWND hEditKeyName, HWND hEditValueName, HWND hEditValueData)

{

int keyNameLength = GetWindowTextLength(hEditKeyName);

if (keyNameLength == 0)

{

MessageBox(nullptr, L"Введите имя ключа", L"Ошибка", MB\_OK);

return;

}

int valueNameLength = GetWindowTextLength(hEditValueName);

if (valueNameLength == 0)

{

MessageBox(nullptr, L"Введите имя данных", L"Ошибка", MB\_OK);

return;

}

int valueDataLength = GetWindowTextLength(hEditValueData);

if (valueDataLength == 0)

{

MessageBox(nullptr, L"Введите значение данных", L"Ошибка", MB\_OK);

return;

}

std::wstring keyName(keyNameLength + 1, L'\0');

std::wstring valueName(valueNameLength + 1, L'\0');

std::wstring valueData(valueDataLength + 1, L'\0');

GetWindowText(hEditKeyName, &keyName[0], keyNameLength + 1);

GetWindowText(hEditValueName, &valueName[0], valueNameLength + 1);

GetWindowText(hEditValueData, &valueData[0], valueDataLength + 1);

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_WRITE, &hParentKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия родительского ключа", L"Ошибка", MB\_OK);

return;

}

HKEY hKey;

if (RegOpenKeyEx(hParentKey, keyName.c\_str(), 0, KEY\_SET\_VALUE, &hKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия ключа", L"Ошибка", MB\_OK);

return;

}

LONG result = RegSetValueEx(hKey, valueName.c\_str(), 0, REG\_SZ, reinterpret\_cast<const BYTE\*>(valueData.c\_str()), (valueDataLength + 1) \* sizeof(wchar\_t));

RegCloseKey(hParentKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Данные успешно изменены", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(nullptr, L"Ошибка при изменении данных", L"Ошибка", MB\_OK);

}

}

void DeleteRegistryValue(HWND hEditKeyName, HWND hEditValueName)

{

int keyNameLength = GetWindowTextLength(hEditKeyName);

if (keyNameLength == 0)

{

MessageBox(nullptr, L"Введите имя ключа", L"Ошибка", MB\_OK);

return;

}

int valueNameLength = GetWindowTextLength(hEditValueName);

if (valueNameLength == 0)

{

MessageBox(nullptr, L"Введите имя значения", L"Ошибка", MB\_OK);

return;

}

std::wstring keyName(keyNameLength + 1, L'\0');

std::wstring valueName(valueNameLength + 1, L'\0');

GetWindowText(hEditKeyName, &keyName[0], keyNameLength + 1);

GetWindowText(hEditValueName, &valueName[0], valueNameLength + 1);

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software", 0, KEY\_WRITE, &hParentKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия родительского ключа", L"Ошибка", MB\_OK);

return;

}

HKEY hKey;

if (RegOpenKeyEx(hParentKey, keyName.c\_str(), 0, KEY\_SET\_VALUE, &hKey) != ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Ошибка открытия ключа", L"Ошибка", MB\_OK);

return;

}

LONG result = RegDeleteValue(hKey, valueName.c\_str());

RegCloseKey(hKey);

if (result == ERROR\_SUCCESS)

{

MessageBox(nullptr, L"Значение успешно удалено", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(nullptr, L"Ошибка при удалении значения", L"Ошибка", MB\_OK);

}

}