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 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc149910353)

[2 Теоретические сведения 4](#_Toc149910354)

[3 Описание функций программы 5](#_Toc149910355)

[3.1 Создание и изменение ключей и их значений 5](#_Toc149910356)

[3.2 Удаление ключей и их значений 5](#_Toc149910357)

[3.3 Получение значений ключа и данных этих значений 6](#_Toc149910358)

[Список использованных источников 7](#_Toc149910359)

[Приложение А 8](#_Toc149910360)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Доступ к реестру Windows. Другие вспомогательные средства управления.

В качестве задачи необходимо разработать утилиту для создания и управления реестровыми записями Windows, включая создание, изменение и удаление ключей и значений.

# **2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Работа с реестром *Windows* в *WinAPI* представляет собой важную часть работы с операционной системой, так как реестр используется для хранения настроек, конфигураций и другой важной информации.

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.[1]

Реестр имеет древовидную структуру, состоящую из ключей (*keys*) и значений (*values*). Каждый ключ может содержать подключи и значения. Основные разделы реестра, такие как *HKEY*\_*LOCAL\_MACHINE*, *HKEY*\_*CURRENT*\_*USER*, *HKEY*\_*CLASSES*\_*ROOT*, и др., являются корневыми ключами.

Для работы с реестром в *WinAPI* используются функции, такие как *RegOpenKeyEx, RegCreateKeyEx, RegSetValueEx, RegQueryValueEx, RegEnumKeyEx* и другие. Эти функции позволяют открывать, создавать, читать, записывать и удалять ключи и значения в реестре.

В реестре хранятся данные различных типов, такие как строки, числа, булевы значения и др. Наиболее распространенные типы данных включают *REG*\_SZ (строки), REG\_*DWORD* (32-битные целые числа), REG\_*QWORD* (64-битные целые числа) и другие.

На 64-битных версиях *Windows* реестр разделен на две ветки: 32-битную и 64-битную. 32-битные приложения имеют свою собственную ветку в реестре, отличную от 64-битных приложений. Это может потребовать учета архитектуры при работе с реестром.[2]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были реализованы следующие функции.

3.1 Создание и изменение ключей и их значений

Пользователю требуется ввести имя ключа в поле для ключа и значение этого ключа в поле для значения, а затем данные значения. Далее нажать кнопку создания. Ключ со значением создается в реестре *Windows* (рисунок 1). Если ключ уже существовал в реестре, то его значение будет обновлено. Также имеется возможность выбрать тип данных который будет использоваться для данных значения.

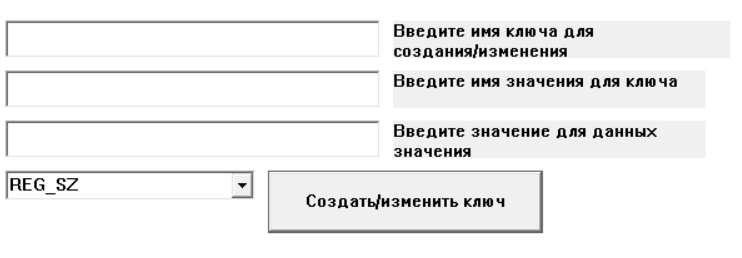


Рисунок 1 – создание или изменение ключа

3.2 Удаление ключей и их значений

Чтобы удалить ключ и его значение, пользователю требуется ввести имя ключа, который нужно удалить. Далее нажать кнопку удаления (рисунок 2).

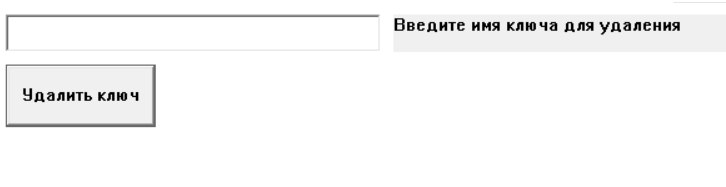


Рисунок 2 – удаление ключа

Чтобы удалить значение, введите имя значения в соответствующее поле, и нажмите кнопку удаления значения (рисунок 3).

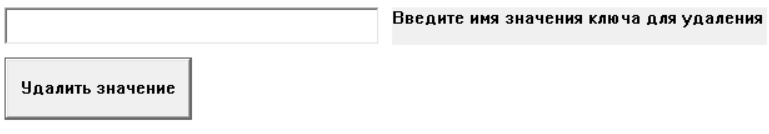


Рисунок 3 – удаление значений ключа

3.3 Получение значений ключа и данных этих значений

Чтобы получить значения ключа и их данные, пользователю требуется ввести имя ключа и нажать найти ключ (рисунок 4).

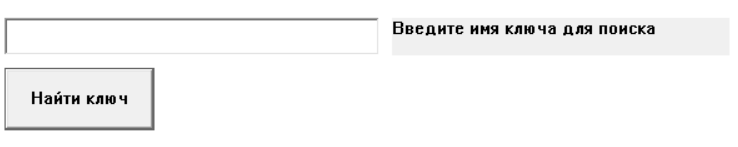


Рисунок 4 – получение данных о ключе

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] Сведения о реестре – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/[about](https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/about-the-registry)-the-registry

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Lab5.h

#pragma once

#include "resource.h"

#include <windows.h>

#include <string>

#define CREATE\_KEY 1101

#define DELETE\_KEY 1102

HWND hEditCreateKeyName;

HWND hEditCreateKeyValue;

HWND hButtonCreateKey;

HWND hEditDeleteKeyName;

HWND hComboBoxType;

HWND hButtonDeleteKey;

bool CreateKey(std::wstring name, std::wstring value);

bool DeleteKey(std::wstring name);

Файл Lab5.cpp

#ifndef UNICODE

#define UNICODE

#endif

#include "Lab5.h"

#include <strsafe.h>

#include <tchar.h>

#include <locale>

#include <codecvt>

#include <atlstr.h>

#include <atlconv.h>

/\*

Разработать утилиту для создания и управления

реестровыми записями Windows, включая создание,

изменение и удаление ключей и значений.

\*/

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

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

{

// Register the window class.

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window.

HWND hwnd = CreateWindowExW(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Learn to Program Windows", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

CW\_USEDEFAULT, CW\_USEDEFAULT, 1300, 700,

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

if (hwnd == NULL)

{

return 0;

}

hEditCreateKeyName = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 10, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndCreateKeyNameText = CreateWindow(

L"STATIC",

L"Введите имя ключа для создания/изменения",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

320, 10, 270, 30,

hwnd,

NULL,

hInstance,

NULL

);

hEditCreateKeyValue = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 50, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndCreateValueText = CreateWindow(

L"STATIC",

L"Введите имя значения для ключа",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

320, 50, 250, 30,

hwnd,

NULL,

hInstance,

NULL

);

hEditCreateKeyValueData = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 90, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndCreateValueDataText = CreateWindow(

L"STATIC",

L"Введите значение для данных значения",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

320, 90, 250, 30,

hwnd,

NULL,

hInstance,

NULL

);

hComboBoxType = CreateWindow(

L"COMBOBOX",

NULL,

WS\_CHILD | WS\_VISIBLE | CBS\_DROPDOWN | CBS\_HASSTRINGS,

10, 130, 200, 200,

hwnd, NULL, hInstance, NULL

);

// Добавление элементов в комбо-бокс

SendMessage(hComboBoxType, CB\_ADDSTRING, 0, (LPARAM)L"REG\_SZ");

SendMessage(hComboBoxType, CB\_ADDSTRING, 0, (LPARAM)L"REG\_DWORD");

SendMessage(hComboBoxType, CB\_ADDSTRING, 0, (LPARAM)L"REG\_QWORD");

// Выбор первого элемента в комбо-боксе

SendMessage(hComboBoxType, CB\_SETCURSEL, 0, 0);

hButtonCreateKey = CreateWindowEx(

0,

L"BUTTON",

L"Создать/изменить ключ",

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

220, 130, 220, 50,

hwnd,

(HMENU)CREATE\_KEY,

hInstance,

NULL

);

hEditDeleteKeyName = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

610, 10, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndDeleteKeyNameText = CreateWindow(

L"STATIC",

L"Введите имя ключа для удаления",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

920, 10, 270, 30,

hwnd,

NULL,

hInstance,

NULL

);

hButtonDeleteKey = CreateWindowEx(

0,

L"BUTTON",

L"Удалить ключ",

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

610, 50, 120, 50,

hwnd,

(HMENU)DELETE\_KEY,

hInstance,

NULL

);

hEditDeleteValueName = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

610, 110, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndDeleteValueNameText = CreateWindow(

L"STATIC",

L"Введите имя значения ключа для удаления",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

920, 110, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

hButtonDeleteValue = CreateWindowEx(

0,

L"BUTTON",

L"Удалить значение",

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

610, 150, 150, 50,

hwnd,

(HMENU)DELETE\_VALUE,

hInstance,

NULL

);

hEditFindKeyName = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 340, 300, 30,

hwnd,

NULL,

hInstance,

NULL

);

HWND hwndFindKeyNameText = CreateWindow(

L"STATIC",

L"Введите имя ключа для поиска",

WS\_VISIBLE | WS\_CHILD | SS\_LEFT,

320, 340, 270, 30,

hwnd,

NULL,

hInstance,

NULL

);

hButtonFindKey = CreateWindowEx(

0,

L"BUTTON",

L"Найти ключ",

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

10, 380, 120, 50,

hwnd,

(HMENU)FIND\_KEY,

hInstance,

NULL

);

ShowWindow(hwnd, nCmdShow);

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

{

switch (uMsg)

{

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

EndPaint(hwnd, &ps);

}

case WM\_COMMAND:

{

int command = LOWORD(wParam);

switch (command)

{

case CREATE\_KEY:

{

LPWSTR name = new WCHAR[GetWindowTextLength(hEditCreateKeyName) + 1];

LPWSTR value = new WCHAR[GetWindowTextLength(hEditCreateKeyValue) + 1];

LPWSTR data = new WCHAR[GetWindowTextLength(hEditCreateKeyValueData) + 1];

GetWindowText(hEditCreateKeyName, name, GetWindowTextLength(hEditCreateKeyName) + 1);

GetWindowText(hEditCreateKeyValue, value, GetWindowTextLength(hEditCreateKeyValue) + 1);

GetWindowText(hEditCreateKeyValueData, data, GetWindowTextLength(hEditCreateKeyValueData) + 1);

if (CreateKey(name, value, data))

{

MessageBox(hwnd, L"Ключ успешно создан в реестре.", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(hwnd, L"Ошибка при создании ключа в реестре.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

delete[] name;

delete[] value;

delete[] data;

break;

}

case DELETE\_KEY:

{

LPWSTR name = new WCHAR[GetWindowTextLength(hEditDeleteKeyName) + 1];

GetWindowText(hEditDeleteKeyName, name, GetWindowTextLength(hEditDeleteKeyName) + 1);

if (DeleteKey(name))

{

MessageBox(hwnd, L"Ключ успешно удален в реестре.", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(hwnd, L"Ошибка при удалении ключа в реестре.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

delete[] name;

break;

}

case DELETE\_VALUE:

{

LPWSTR name = new WCHAR[GetWindowTextLength(hEditDeleteValueName) + 1];

GetWindowText(hEditDeleteValueName, name, GetWindowTextLength(hEditDeleteValueName) + 1);

if (DeleteValue(name))

{

MessageBox(hwnd, L"Значение успешно удален в реестре.", L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

else

{

MessageBox(hwnd, L"Ошибка при удалении значения в реестре.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

delete[] name;

break;

}

case FIND\_KEY:

{

LPWSTR name = new WCHAR[GetWindowTextLength(hEditFindKeyName) + 1];

GetWindowText(hEditFindKeyName, name, GetWindowTextLength(hEditFindKeyName) + 1);

std::wstring result = FindKey(name);

if (result.empty())

{

MessageBox(hwnd, L"Ключ не был найден в реестре.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

else

{

std::wstring mywstring(name);

result = L"Ключ с именем " + mywstring + L" имеет данные:\n" + result;

MessageBox(hwnd, result.c\_str(), L"Успех", MB\_OK | MB\_ICONINFORMATION);

}

delete[] name;

break;

}

}

}

return 0;

}

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

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// RegDelnodeRecurse()

//

// Purpose: Deletes a registry key and all its subkeys / values.

//

// Parameters: hKeyRoot - Root key

// lpSubKey - SubKey to delete

//

// Return: TRUE if successful.

// FALSE if an error occurs.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL RegDelnodeRecurse(HKEY hKeyRoot, LPTSTR lpSubKey)

{

LPTSTR lpEnd;

LONG lResult;

DWORD dwSize;

TCHAR szName[MAX\_PATH];

HKEY hKey;

FILETIME ftWrite;

// First, see if we can delete the key without having

// to recurse.

lResult = RegDeleteKey(hKeyRoot, lpSubKey);

if (lResult == ERROR\_SUCCESS)

return TRUE;

lResult = RegOpenKeyEx(hKeyRoot, lpSubKey, 0, KEY\_READ, &hKey);

if (lResult != ERROR\_SUCCESS)

{

if (lResult == ERROR\_FILE\_NOT\_FOUND) {

printf("Key not found.\n");

return TRUE;

}

else {

printf("Error opening key.\n");

return FALSE;

}

}

// Check for an ending slash and add one if it is missing.

lpEnd = lpSubKey + lstrlen(lpSubKey);

if (\*(lpEnd - 1) != TEXT('\\'))

{

\*lpEnd = TEXT('\\');

lpEnd++;

\*lpEnd = TEXT('\0');

}

// Enumerate the keys

dwSize = MAX\_PATH;

lResult = RegEnumKeyEx(hKey, 0, szName, &dwSize, NULL,

NULL, NULL, &ftWrite);

if (lResult == ERROR\_SUCCESS)

{

do {

\*lpEnd = TEXT('\0');

StringCchCat(lpSubKey, MAX\_PATH \* 2, szName);

if (!RegDelnodeRecurse(hKeyRoot, lpSubKey)) {

break;

}

dwSize = MAX\_PATH;

lResult = RegEnumKeyEx(hKey, 0, szName, &dwSize, NULL,

NULL, NULL, &ftWrite);

} while (lResult == ERROR\_SUCCESS);

}

lpEnd--;

\*lpEnd = TEXT('\0');

RegCloseKey(hKey);

// Try again to delete the key.

lResult = RegDeleteKey(hKeyRoot, lpSubKey);

if (lResult == ERROR\_SUCCESS)

return TRUE;

return FALSE;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// RegDelnode()

//

// Purpose: Deletes a registry key and all its subkeys / values.

//

// Parameters: hKeyRoot - Root key

// lpSubKey - SubKey to delete

//

// Return: TRUE if successful.

// FALSE if an error occurs.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BOOL RegDelnode(HKEY hKeyRoot, LPCTSTR lpSubKey)

{

TCHAR szDelKey[MAX\_PATH \* 2];

StringCchCopy(szDelKey, MAX\_PATH \* 2, lpSubKey);

return RegDelnodeRecurse(hKeyRoot, szDelKey);

}

std::string ws2s(const std::wstring& wstr)

{

using convert\_typeX = std::codecvt\_utf8<wchar\_t>;

std::wstring\_convert<convert\_typeX, wchar\_t> converterX;

return converterX.to\_bytes(wstr);

}

bool CreateKey(std::wstring keyName, std::wstring value, std::wstring data)

{

if (value.empty())

{

return false;

}

HKEY hKey;

LPCWSTR keyPath;

TCHAR\* dst = new TCHAR[keyName.size()];

if (!keyName.empty())

{

//setup converter

using convert\_type = std::codecvt\_utf8<wchar\_t>;

std::wstring\_convert<convert\_type, wchar\_t> converter;

//use converter (.to\_bytes: wstr->str, .from\_bytes: str->wstr)

std::string converted\_str = converter.to\_bytes(keyName);

converted\_str = "Software\\Lab5\\" + converted\_str;

//As much as we'd love to, we can't use memcpy() because

//sizeof(TCHAR)==sizeof(char) may not be true:

std::copy(converted\_str.begin(), converted\_str.end(), dst);

dst[converted\_str.size()] = '\0';

}

else

{

std::string str = "Software\\Lab5";

std::copy(str.begin(), str.end(), dst);

dst[str.size()] = '\0';

}

LPCWSTR valueName = value.c\_str();

LPCWSTR valueData = data.c\_str();

int selectedIndex = SendMessage(hComboBoxType, CB\_GETCURSEL, 0, 0);

if (selectedIndex == CB\_ERR)

{

return false;

}

LONG result = RegCreateKeyEx(HKEY\_CURRENT\_USER, dst, 0, NULL, REG\_OPTION\_NON\_VOLATILE, KEY\_WRITE, NULL, &hKey, NULL);

if (result == ERROR\_SUCCESS)

{

if (selectedIndex == 0)

{

result = RegSetValueEx(hKey, valueName, 0, REG\_SZ, (const BYTE\*)valueData, (wcslen(valueData) + 1) \* sizeof(WCHAR));

if (result != ERROR\_SUCCESS)

{

return false;

}

RegCloseKey(hKey);

return true;

}

if (selectedIndex == 1)

{

DWORD dwValue = wcstoul(valueData, NULL, 10);

result = RegSetValueEx(hKey, valueName, 0, REG\_DWORD, (const BYTE\*)&dwValue, sizeof(DWORD));

if (result != ERROR\_SUCCESS)

{

return false;

}

RegCloseKey(hKey);

return true;

}

if (selectedIndex == 2)

{

ULONGLONG qwValue = \_wcstoui64(valueData, NULL, 10);

result = RegSetValueEx(hKey, valueName, 0, REG\_QWORD, (const BYTE\*)&qwValue, sizeof(ULONGLONG));

if (result != ERROR\_SUCCESS)

{

return false;

}

RegCloseKey(hKey);

return true;

}

}

else {

return false;

}

}

bool DeleteValue(std::wstring name)

{

if (name.empty())

{

return false;

}

HKEY hKey = NULL;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, \_T("Software\\Lab5"), 0L, KEY\_SET\_VALUE, &hKey) == ERROR\_SUCCESS)

{

TCHAR\* dst = W2T((LPWSTR)name.c\_str());

if (RegDeleteValue(hKey, dst) == ERROR\_SUCCESS)

{

return true;

}

}

else

{

return false;

}

}

bool DeleteKey(std::wstring name)

{

HKEY hKey;

if (name.empty())

{

return false;

}

std::wstring new\_name(L"Software\\Lab5\\");

new\_name += name;

return RegDelnode(HKEY\_CURRENT\_USER, new\_name.c\_str());

}

std::wstring FindKey(std::wstring name)

{

HKEY hKey = HKEY\_CURRENT\_USER; // Корневой ключ, в котором находится ключ

std::wstring keyName = L"Software\\Lab5"; // Путь к ключу

std::wstring s1 = std::wstring(keyName);

std::wstring s2 = std::wstring(L"\\");

std::wstring s3 = std::wstring(name);

std::wstring subkeyName = (s1 + s2 + s3);

LONG result = RegOpenKeyExW(hKey, subkeyName.c\_str(), 0, KEY\_READ, &hKey);

if (result == ERROR\_SUCCESS) {

// Открытие ключа прошло успешно

DWORD valueIndex = 0; // Индекс значения

WCHAR valueName[256]; // Максимальная длина имени значения

DWORD valueNameSize = 256;

DWORD dataType;

BYTE data[256]; // Максимальный размер данных значения

DWORD dataSize = 256;

std::wstring answer;

while (true) {

result = RegEnumValueW(hKey, valueIndex, valueName, &valueNameSize, NULL, &dataType, data, &dataSize);

if (result == ERROR\_NO\_MORE\_ITEMS) {

// Все значения перечислены

break;

}

if (result == ERROR\_SUCCESS) {

answer += L"Имя значения: " + std::wstring(valueName) + L'\n';

if (dataType == REG\_SZ) {

LPCWSTR strValue = reinterpret\_cast<LPCWSTR>(data);

answer += L"Данные значения: " + std::wstring(strValue) + L'\n';

}

else if (dataType == REG\_DWORD) {

DWORD intValue = \*reinterpret\_cast<DWORD\*>(data);

answer += L"Данные значения: " + std::wstring(std::to\_wstring(intValue)) + L'\n';

}

else {

answer += L"Другой тип данных (не обработан).\n";

}

// Увеличение индекса и сброс размеров

valueIndex++;

valueNameSize = 256;

dataSize = 256;

answer += L'\n';

}

else {

wprintf(L"Ошибка при перечислении значений: %d\n", result);

break;

}

}

return answer;

RegCloseKey(hKey);

}

else {

wprintf(L"Ошибка при открытии ключа: %d\n", result);

}

//HKEY hKey = HKEY\_CURRENT\_USER;

//std::wstring subkeyName = L"Software\\Lab5";

//DWORD dataType;

//BYTE\* data = nullptr;

//DWORD dataSize = 0;

//if (RegOpenKeyExW(hKey, subkeyName.c\_str(), 0, KEY\_READ, &hKey) == ERROR\_SUCCESS)

//{

// long result = RegQueryValueExW(hKey, (LPCWSTR)name.c\_str(), NULL, &dataType, NULL, &dataSize);

// if (result == ERROR\_SUCCESS) {

// // Выделение памяти для данных значения

// data = new BYTE[dataSize];

// result = RegQueryValueExW(hKey, (LPCWSTR)name.c\_str(), NULL, &dataType, data, &dataSize);

// if (result == ERROR\_SUCCESS) {

// if (dataType == REG\_SZ) {

// // Если тип данных - строка (REG\_SZ)

// LPCWSTR strValue = reinterpret\_cast<LPCWSTR>(data);

// return strValue;

// }

// else if (dataType == REG\_DWORD) {

// // Если тип данных - 32-битное целое число (REG\_DWORD)

// DWORD intValue = \*reinterpret\_cast<DWORD\*>(data);

// return std::wstring(std::to\_wstring(intValue));

// }

// else {

// wprintf(L"Другой тип данных (не обработан).\n");

// }

// }

// else {

// wprintf(L"Ошибка при получении значения: %d\n", result);

// }

// }

// else {

// wprintf(L"Ошибка при получении информации о значении: %d\n", result);

// }

//}

//return L"";

}