

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 Миненков Я.А.

Проверил ассистент кафедры информатики
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Формулировка задачи	3
2 Теоретические сведения	4
3 Описание функций программы.....	5
Список использованных источников	7
Приложение А (обязательное) Листинг кода.....	8

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является разработка утилиты для создания и управления реестровыми записями *Windows*, включая создание, изменение и удаление ключей и значений.

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 введен для упорядочения информации, хранившейся до этого во множестве INI-файлов. [1]

Для работы с реестром *Windows* были использованы следующие функции: *RegOpenKeyEx* — для открытия ключа в реестре, *RegQueryValueEx* — для получения значения из реестра, *RegCloseKey* — для закрытия открытого ключа реестра. *RegCreateKeyEx* — для создания нового ключа реестра или открытия существующего, *RegDeleteKeyW* — для удаления ключа реестра, *RegSetValueExA* — для установки значения в реестр. [2]

Значение реестра может хранить данные в одном из нескольких форматов, например, строковое (REG_SZ — строка с нулевым символом в конце) или целочисленное значение (REG_DWORD — 32-разрядное число, REG_QWORD — 64-разрядное число). При хранении данных в значении реестра, например, путем вызова функции *RegSetValueEx*, можно указать тип хранимых данных, указав один из типов в таблице ниже. При получении значения реестра такие функции, как *RegQueryValueEx*, используют эти типы для указания типа полученных данных.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

При запуске приложения пользователь имеет возможность создать ключ с помощью кнопки “Создать ключ” или же удалить с помощью кнопки “Удалить ключ” (Рисунок 1).

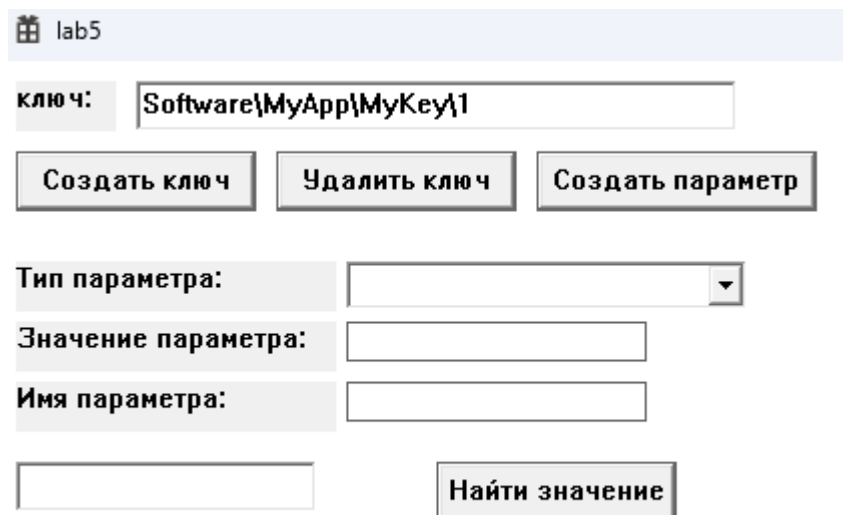


Рисунок 1 – Главное окно приложения

Также можно выполнить поиск существующего значения (Рисунок 2).

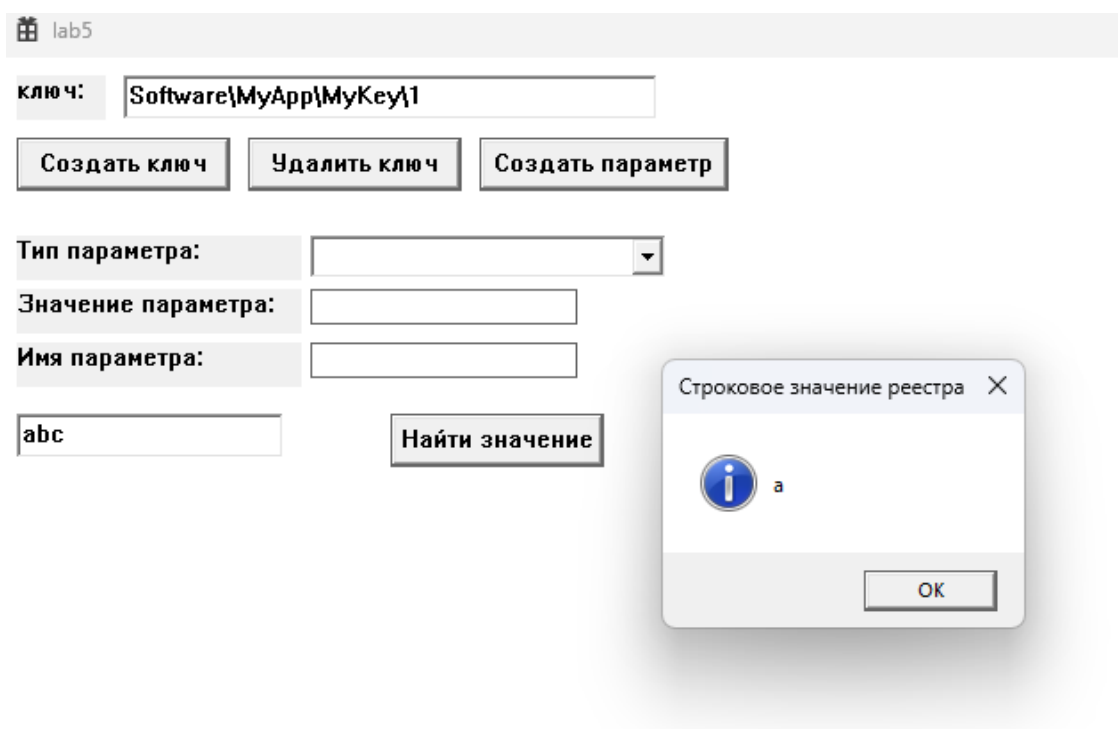


Рисунок 2 – Результат выполнения поиска

Пользователь может добавить значение в существующий ключ, указав тип параметра, значение и имя параметра (Рисунок 3).

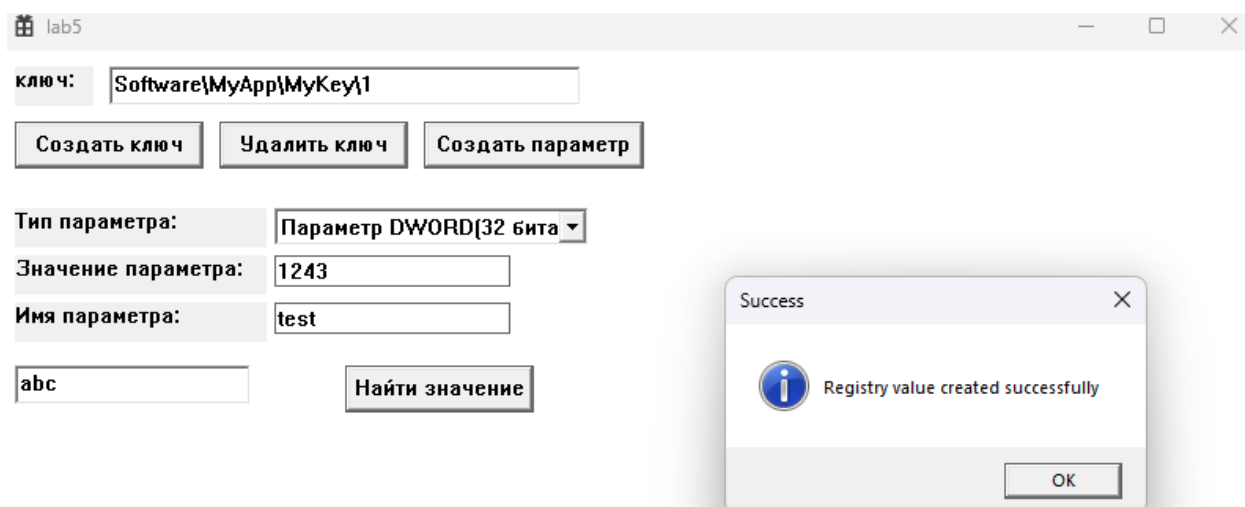


Рисунок 3 – Создание параметра в ключе реестра

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Реестр Windows [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Реестр_Windows
- [2] Функции реестра – *Win32 apps* [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry-functions>
- [3] *Regisrty value types* – *Win32 apps* [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry-value-types>

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

Листинг 1 – Файл lab5.cpp

```
#include "framework.h"
#include "lab5.h"
#include <stdio.h>
#include <Windows.h>
#include <string>
#include <vector>

#define IDC_EDIT_SUBKEY 1101
#define IDC_BUTTON_CREATE_KEY 1102
#define IDC_BUTTON_DELETE_KEY 1103
#define IDC_BUTTON_ADD_NEW_PARAMETER 1104
#define IDC_BUTTON_SEARCH_VALUE 1105

#define MAX_LOADSTRING 100

HWND hWnd;
HMENU hMenu;
HWND hEdit;
HWND hButton, hButtonDelete, hButtonAddNewParameter;
HWND hEditValue;

HWND hComboBox;
HWND hValueDataEdit;
HWND hValueNameEdit;

HWND hMainWindow, hValueList;
HWND hSubKeyEdit;

struct RegistryValue
{
    std::wstring name;
    DWORD type;
    std::wstring data;
};

bool CreateRegistryKey(HKEY hRootKey, LPCWSTR subKey);
void PopulateValueList(HKEY hKey, LPCWSTR subKey);
void LoadRegistryInfoToListBox(HWND hListBox, LPCWSTR subKey);
bool DeleteRegistryKey(HKEY hKey, LPCWSTR subKey);
bool SetRegistryValue(HKEY hKey, LPCSTR subKey, LPCSTR valueName, DWORD
valueType, const BYTE* valueData, DWORD dataSize);
bool CreateRegistryValue();
void getRegistryValue(LPCTSTR subKey, LPCTSTR name);

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
```



```

        _In_opt_ HINSTANCE hPrevInstance,
        _In_ LPWSTR lpCmdLine,
        _In_ int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB5, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB5));

    MSG msg;

    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB5));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH) (COLOR_WINDOW+1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB5);
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    const wchar_t CLASS_NAME[] = L"RegistryEditor";

    hInst = hInstance;

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,

```

```

        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance,
        nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

    HWND hStatic = CreateWindowEx(
        0,
        L"STATIC",
        L"ключ:",
        WS_CHILD | WS_VISIBLE,
        10, 10, 50, 25,
        hWnd,
        NULL,
        hInstance,
        NULL
    );

    hEdit = CreateWindowEx(
        WS_EX_CLIENTEDGE,
        L"EDIT",
        L"",
        WS_CHILD | WS_VISIBLE | ES_AUTOHSCROLL,
        70, 10, 300, 25,
        hWnd,
        (HMENU) IDC_EDIT_SUBKEY,
        hInstance,
        NULL
    );

    hButton = CreateWindowEx(
        0,
        L"BUTTON",
        L"Создать ключ",
        WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
        10, 45, 120, 30,
        hWnd,
        (HMENU) IDC_BUTTON_CREATE_KEY,
        hInstance,
        NULL
    );

    hButtonDelete = CreateWindowEx(
        0,
        L"BUTTON",
        L"Удалить ключ",
        WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
        140, 45, 120, 30,
        hWnd,
        (HMENU) IDC_BUTTON_DELETE_KEY,
        hInstance,
        NULL
    );

    hButtonAddNewParameter = CreateWindowEx(
        0,
        L"BUTTON",
        L"Создать параметр",
        WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
        270, 45, 140, 30,
        hWnd,

```

```

        (HMENU) IDC_BUTTON_ADD_NEW_PARAMETER,
        hInstance,
        NULL
    );

HWND hStaticParamType = CreateWindow(
    L"STATIC",
    L"Тип параметра:",
    WS_CHILD | WS_VISIBLE,
    10, 100, 160, 25,
    hWnd,
    NULL,
    hInstance,
    NULL
);

HWND hStaticValueData = CreateWindow(
    L"STATIC",
    L"Значение параметра:",
    WS_CHILD | WS_VISIBLE,
    10, 130, 160, 25,
    hWnd,
    NULL,
    hInstance,
    NULL
);

HWND hStaticValueName = CreateWindow(
    L"STATIC",
    L"Имя параметра:",
    WS_CHILD | WS_VISIBLE,
    10, 160, 160, 25,
    hWnd,
    NULL,
    hInstance,
    NULL
);

hEditValue = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    L"EDIT",
    L"",
    WS_CHILD | WS_VISIBLE | ES_AUTOHSCROLL,
    10, 200, 150, 25,
    hWnd,
    (HMENU) IDC_EDIT_SUBKEY,
    hInstance,
    NULL
);

HWND hButtonSearch = CreateWindowEx(
    0,
    L"BUTTON",
    L"Найти значение",
    WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
    220, 200, 120, 30,
    hWnd,
    (HMENU) IDC_BUTTON_SEARCH_VALUE,
    hInstance,
    NULL
);

hMenu = CreateMenu();

```

```

HMENU hSubMenu = CreatePopupMenu();
SetMenu(hWnd, hMenu);

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
        {
            hComboBox = CreateWindow(TEXT("ComboBox"), NULL, WS_VISIBLE | WS_CHILD
| CBS_DROPDOWNLIST,
            175, 100, 200, 200, hWnd, NULL, NULL, NULL);

            SendMessage(hComboBox, CB_ADDSTRING, 0,
reinterpret_cast<LPARAM>(TEXT("Строковый параметр")));
            SendMessage(hComboBox, CB_SETITEMDATA, 0, REG_SZ);
            SendMessage(hComboBox, CB_ADDSTRING, 0,
reinterpret_cast<LPARAM>(TEXT("Параметр DWORD(32 бита)")));
            SendMessage(hComboBox, CB_SETITEMDATA, 1, REG_DWORD);
            SendMessage(hComboBox, CB_ADDSTRING, 0,
reinterpret_cast<LPARAM>(TEXT("Параметр QWORD(64 бита)")));
            SendMessage(hComboBox, CB_SETITEMDATA, 2, REG_QWORD);

            hValueDataEdit = CreateWindow(TEXT("EDIT"), NULL, WS_VISIBLE | WS_CHILD
| WS_BORDER,
            175, 130, 150, 20, hWnd, NULL, NULL, NULL);

            hValueNameEdit = CreateWindow(TEXT("EDIT"), NULL, WS_VISIBLE | WS_CHILD
| WS_BORDER,
            175, 160, 150, 20, hWnd, NULL, NULL, NULL);

        }
        break;
        case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            switch (wmId)
            {
                case IDC_BUTTON_CREATE_KEY:
                {
                    int length = GetWindowTextLength(hEdit);
                    LPWSTR subKey = new WCHAR[length + 1];
                    GetWindowText(hEdit, subKey, length + 1);
                    if (CreateRegistryKey(HKEY_CURRENT_USER, subKey))
                    {
                        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[] subKey;
                }
            }
            break;
        }
    }
}

```

```

case IDC_BUTTON_SEARCH_VALUE:
{
    int length = GetWindowTextLength(hEditValue);
    TCHAR* buffer = new TCHAR[length + 1];

    GetWindowText(hEditValue, buffer, length + 1);

    int length2 = GetWindowTextLength(hEdit);
    TCHAR* buffer2 = new TCHAR[length2 + 1];

    GetWindowText(hEdit, buffer2, length2 + 1);

    getRegistryValue(buffer2, buffer);
}
break;
case IDC_BUTTON_DELETE_KEY:
{
    int length = GetWindowTextLength(hEdit);
    LPWSTR subKey = new WCHAR[length + 1];
    GetWindowText(hEdit, subKey, length + 1);

    HKEY hKey;
    if (RegOpenKeyExW(HKEY_CURRENT_USER, NULL, 0, KEY_ALL_ACCESS,
&hKey) == ERROR_SUCCESS) {
        if (DeleteRegistryKey(hKey, subKey)) {
            MessageBox(hWnd, L"Ключ реестра успешно удален.", L"Успех",
MB_OK | MB_ICONINFORMATION);
        }
        else {
            MessageBox(hWnd, L"Не удалось удалить ключ реестра.",
L"Ошибка", MB_OK | MB_ICONINFORMATION);
        }

        RegCloseKey(hKey);
    }
    else {
        MessageBox(hWnd, L"Не удалось открыть ключ реестра.",
L"Ошибка", MB_OK | MB_ICONINFORMATION);
    }

    delete[] subKey;
}
break;
case IDC_BUTTON_ADD_NEW_PARAMETER:
{
    CreateRegistryValue();
}
break;
}
break;
}

case WM_PAINT:
{
    PAINTSTRUCT ps;
    HDC hdc = BeginPaint(hWnd, &ps);

    EndPaint(hWnd, &ps);
}
break;
case WM_DESTROY:

```

```

        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

bool CreateRegistryKey(HKEY hRootKey, LPCWSTR subKey)
{
    HKEY hKey;
    LONG result;

    result = RegCreateKeyEx(hRootKey, subKey, 0, NULL, REG_OPTION_NON_VOLATILE,
KEY_ALL_ACCESS, NULL, &hKey, NULL);
    if (result == ERROR_SUCCESS)
    {
        RegCloseKey(hKey);
        return true;
    }
    else
    {
        return false;
    }
}

bool DeleteRegistryKey(HKEY hKey, LPCWSTR subKey) {
    if (RegDeleteKeyW(hKey, subKey) == ERROR_SUCCESS) {
        return true;
    }
    else {
        return false;
    }
}

bool SetRegistryValue(HKEY hKey, LPCSTR subKey, LPCSTR valueName, DWORD
valueType, const BYTE* valueData, DWORD dataSize)
{
    HKEY hSubKey;
    LONG openResult = RegCreateKeyExA(hKey, subKey, 0, NULL,
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hSubKey, NULL);

    if (openResult != ERROR_SUCCESS) {
        MessageBox(hWnd, L"Failed to create registry key.", L"Ycnex", MB_OK |
MB_ICONINFORMATION);
        return false;
    }

    LONG setValueResult = RegSetValueExA(hSubKey, valueName, 0, valueType,
valueData, dataSize);

    if (setValueResult != ERROR_SUCCESS) {
        MessageBox(hWnd, L"Failed to set registry value", L"Ycnex", MB_OK |
MB_ICONINFORMATION);
        RegCloseKey(hSubKey);
        return false;
    }

    RegCloseKey(hSubKey);
    return true;
}

bool CreateRegistryValue()

```

```

{
    int selectedIndex = SendMessage(hComboBox, CB_GETCURSEL, 0, 0);
    DWORD valueType = static_cast<DWORD>(SendMessage(hComboBox,
CB_GETITEMDATA, selectedIndex, 0));

    char valueData[256];
    GetWindowTextA(hValueDataEdit, valueData, sizeof(valueData));

    char valueName[256];
    GetWindowTextA(hValueNameEdit, valueName, sizeof(valueName));

    HKEY hKey;
    int length = GetWindowTextLength(hEdit);
    //LPWSTR subKey = new WCHAR[length + 1];
    //GetWindowText(hEdit, subKey, length + 1);
    LPWSTR buffer = new WCHAR[length + 1];
    GetWindowText(hEdit, buffer, length + 1);
    //LPCSTR subKey = "Software\\MyApp";
    LONG openResult = RegCreateKeyEx(HKEY_CURRENT_USER, buffer, 0, NULL,
REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, NULL, &hKey, NULL);

    if (openResult != ERROR_SUCCESS) {
        MessageBox(NULL, L"Failed to create registry key", L"Error",
MB_ICONERROR);
        return FALSE;
    }

    LONG setValueResult;
    if (valueType == 4) {

        DWORD result;
        char* endPtr;

        result = strtoul(valueData, &endPtr, 10);

        if (*endPtr == '\\0') {
        }
        else {
        }
        BYTE data[sizeof(DWORD)];
        memcpy(data, &result, sizeof(DWORD));
        setValueResult = RegSetValueExA(hKey, valueName, 0, REG_DWORD, data,
sizeof(DWORD));
    }
    else if (valueType == 1) {
        setValueResult = RegSetValueExA(hKey, valueName, 0, valueType,
reinterpret_cast<const BYTE*>(valueData), strlen(valueData) + 1);
    }

    if (setValueResult != ERROR_SUCCESS) {
        MessageBox(NULL, L"Failed to set registry value", L"Error",
MB_ICONERROR);
        RegCloseKey(hKey);
        return FALSE;
    }

    RegCloseKey(hKey);
    MessageBox(NULL, L"Registry value created successfully", L"Success",
MB_ICONINFORMATION);

    return TRUE;
}

```

```

}

void DeleteSubkeys(HKEY hKey) {
    TCHAR achKey[255];
    DWORD cbName;
    LONG lResult;

    while (true) {
        cbName = 255;
        lResult = RegEnumKeyEx(hKey, 0, achKey, &cbName, NULL, NULL, NULL,
NULL);
        if (lResult == ERROR_NO_MORE_ITEMS) {
            break;
        }
        if (lResult == ERROR_SUCCESS) {
            HKEY hSubKey;
            if (RegOpenKeyEx(hKey, achKey, 0, KEY_ALL_ACCESS, &hSubKey) ==
ERROR_SUCCESS) {
                // Рекурсивно удаляем подключи
                DeleteSubkeys(hSubKey);
                RegCloseKey(hSubKey);
            }
        }
    }
}

void getRegistryValue(LPCTSTR subKey, LPCTSTR name)
{
    HKEY hKey;
    DWORD dwType = 0;
    BYTE data[1024];
    DWORD dwDataSize = sizeof(data);
    LONG lReturn;

    lReturn = RegOpenKeyEx(HKEY_CURRENT_USER, subKey, 0, KEY_READ, &hKey);

    if (lReturn == ERROR_SUCCESS)
    {
        lReturn = RegQueryValueEx(hKey, name, NULL, &dwType, data,
&dwDataSize);

        if (lReturn == ERROR_SUCCESS)
        {
            switch (dwType)
            {
                case REG_SZ:
                    MessageBoxA(NULL, reinterpret_cast<const char*>(data),
"Строковое значение реестра", MB_OK | MB_ICONINFORMATION);
                    break;
                case REG_DWORD:
                    char buffer[32];
                    sprintf_s(buffer, sizeof(buffer), "%lu",
*reinterpret_cast<DWORD*>(data));
                    MessageBoxA(NULL, buffer, "DWORD значение реестра", MB_OK |
MB_ICONINFORMATION);
                    break;
                default:
                    MessageBoxA(NULL, "Неизвестный тип данных.", "Ошибка", MB_OK |
MB_ICONERROR);
            }
        }
        else
        {

```



```
        MessageBoxA(NULL, "Не могу получить значение параметра из  
реестра.", "Ошибка", MB_OK | MB_ICONERROR);  
    }  
  
    RegCloseKey(hKey);  
}  
else  
{  
    MessageBoxA(NULL, "Не могу открыть ключ реестра.", "Ошибка", MB_OK |  
MB_ICONERROR);  
}  
}
```