Системное программирование

Дедов Никита, 251003

Лабораторная №3

#include <windows.h>

#include <stdio.h>

#define BUTTON\_WIDTH 100

#define BUTTON\_HEIGHT 30

#define MAX\_KEYS 256

#define MAX\_VALUES 256

#define MAX\_KEY\_LENGTH 255

#define MAX\_VALUE\_NAME 255

#define ID\_OPEN\_KEY 1

#define ID\_READ\_VALUE 2

#define ID\_SET\_VALUE 3

#define ID\_CREATE\_KEY 4

#define ID\_DELETE\_KEY 5

#define ID\_SET\_DATA 6

#define ID\_BACK 7

#define ID\_CREATE\_VALUE 8

#define ID\_DELETE\_VALUE 9

#define ID\_READ\_TYPE 10

#define IDC\_LISTBOX\_KEYS 11

#define IDC\_LISTBOX\_VALUES 12

#define IDC\_MY\_COMBOBOX 13

#define ID\_EDIT 14

#define ID\_LABEL\_KEY 15

#define ID\_LABEL\_VALUE 16

#define ACTION\_CREATE\_KEY 17

#define ACTION\_SET\_VALUE 18

#define ACTION\_CREATE\_VALUE 19

typedef struct{

char\* btnName;

int btnId;

} buts;

buts btnInfo[] = {

{"Open Key", ID\_OPEN\_KEY},

{"Back", ID\_BACK},

{"Create Key", ID\_CREATE\_KEY},

{"Delete Key", ID\_DELETE\_KEY},

{"Create Value", ID\_CREATE\_VALUE},

{"Read Value", ID\_READ\_VALUE},

{"Set Value", ID\_SET\_VALUE},

{"Delete Value", ID\_DELETE\_VALUE},

{"Read Type", ID\_READ\_TYPE},

{NULL, 0}

};

typedef struct {

char name[MAX\_KEY\_LENGTH];

} KeyInfo;

typedef struct {

char name[MAX\_VALUE\_NAME];

BYTE data[MAX\_VALUE\_NAME];

DWORD dataType;

} ValueInfo;

const char\* subKey = "";

char valueName[MAX\_KEY\_LENGTH] = { "\0" };

char newValue[MAX\_KEY\_LENGTH] = { "\0" };

char chosenFile[MAX\_KEYS][MAX\_KEY\_LENGTH] = {'\0'};

HKEY hKeys[MAX\_KEYS] = { HKEY\_CURRENT\_USER };

int pointer = 0;

DWORD subKeyCount, valueCount;

KeyInfo keys[MAX\_KEYS];

ValueInfo values[256];

int action = 0;

static HWND hListBoxKeys, hListBoxValues;

static HWND hEdit, hBtnOk, hComboBox;

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

int OpenRegistryKeyForFirst(HWND hwnd);

int OpenRegistryKey(HWND hwnd, const char\* keyName, int i);

void ReadRegistryValue(HWND hwnd, const char\* valueName);

void SetRegistryValue(HWND hwnd, const char\* valueName, const char\* newValue, DWORD type);

void DeleteRegistryValue(HWND hwnd, const char\* valueName);

void CreateRegistryKey(HWND hwnd, const char\* valueName);

void DeleteRegistryKey(HWND hwnd, const char\* valueName);

void ShowRegistryKeysAndValues(HWND hwnd);

void FillDataAboutKey(int i);

void ReadRegistryType(HWND hwnd, const char\* valueName);

const char\* GetRegTypeName(DWORD dataType);

DWORD GetRegType(const char\* type);

int GetIdOfType(DWORD type);

DWORD GetType(HWND hwnd, const char\* valueName);

char\* GetData(HWND hwnd, const char\* valueName);

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX wcex; HWND hWnd; MSG msg;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_DBLCLKS;

wcex.lpfnWndProc = WindowProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = NULL;

wcex.lpszClassName = "Lab2";

wcex.hIconSm = wcex.hIcon;

RegisterClassEx(&wcex);

hWnd = CreateWindow("Lab2", L"Second lab",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0,

CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}

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

switch (uMsg) {

case WM\_ERASEBKGND: {

HDC hdc = (HDC)wParam;

RECT rect;

GetClientRect(hwnd, &rect);

HBRUSH hBrush = CreateSolidBrush(RGB(0, 0, 255));

FillRect(hdc, &rect, hBrush);

DeleteObject(hBrush);

return 1;

}

case WM\_CTLCOLORSTATIC: {

HDC hdcStatic = (HDC)wParam;

HWND hWndStatic = (HWND)lParam;

SetBkMode(hdcStatic, TRANSPARENT);

SetTextColor(hdcStatic, RGB(255, 255, 0));

return (LRESULT)GetStockObject(NULL\_BRUSH);

}

case WM\_CREATE: {

int i = 0;

for (i = 0; btnInfo[i].btnName != NULL; i++) {

CreateWindowA("BUTTON", btnInfo[i].btnName,

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

10 \* (i + 1) + BUTTON\_WIDTH \* i, 10, BUTTON\_WIDTH, BUTTON\_HEIGHT,

hwnd, (HMENU)btnInfo[i].btnId, NULL, NULL);

}

hEdit = CreateWindowA("EDIT", "",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER,

50 + 10 \* (i + 1) + BUTTON\_WIDTH \* i, 10, BUTTON\_WIDTH, 30,

hwnd, (HMENU)ID\_EDIT, NULL, NULL);

ShowWindow(hEdit, SW\_HIDE);

hComboBox = CreateWindowA("COMBOBOX", "", WS\_CHILD | WS\_VISIBLE | CBS\_DROPDOWNLIST,

50 + 10 \* (i + 1) + BUTTON\_WIDTH \* i, 50, 200, 150 / 9 \* 7,

hwnd, (HMENU)IDC\_MY\_COMBOBOX, NULL, NULL);

ShowWindow(hComboBox, SW\_HIDE);

char\* items[] = { "REG\_SZ", "REG\_DWORD", "REG\_BINARY", "REG\_EXPAND\_SZ", "REG\_MULTI\_SZ", NULL };

for (int j = 0; items[j] != NULL; ++j) {

SendMessageA(hComboBox, CB\_ADDSTRING, 0, (LPARAM)items[j]);

}

SendMessageA(hComboBox, CB\_SETCURSEL, 0, 0);

i++;

hBtnOk = CreateWindowA("BUTTON", "OK",

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

50 + 10 \* (i + 1) + BUTTON\_WIDTH \* i, 10, BUTTON\_WIDTH, BUTTON\_HEIGHT,

hwnd, (HMENU)ID\_SET\_DATA, NULL, NULL);

ShowWindow(hBtnOk, SW\_HIDE);

i++;

CreateWindowA("static", " Keys",

WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP,

10, 50, 300, 20,

hwnd, (HMENU)ID\_LABEL\_KEY, (HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

hListBoxKeys = CreateWindowA("LISTBOX", "Keys", WS\_CHILD | WS\_VISIBLE | LBS\_NOTIFY,

10, 90, 300, 150,

hwnd, (HMENU)IDC\_LISTBOX\_KEYS,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

CreateWindowA("static", " Values",

WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP,

320, 50, 300, 20,

hwnd, (HMENU)ID\_LABEL\_VALUE, NULL, NULL);

hListBoxValues = CreateWindowA("LISTBOX", "Values", WS\_CHILD | WS\_VISIBLE | LBS\_NOTIFY,

320, 90, 200, 150,

hwnd, (HMENU)IDC\_LISTBOX\_VALUES,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE), NULL);

OpenRegistryKeyForFirst(hwnd);

break;

}

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case ID\_OPEN\_KEY: {

pointer -= OpenRegistryKey(hwnd, chosenFile[pointer], 1);

pointer++;

valueName[0] = '\0';

break;

}

case ID\_READ\_VALUE: {

ReadRegistryValue(hwnd, valueName);

break;

}

case ID\_SET\_VALUE: {

SendMessageA(hComboBox, CB\_SETCURSEL, GetIdOfType(GetType(hwnd, valueName)), 0);

SetWindowTextA(hEdit, GetData(hwnd, valueName));

ShowWindow(hEdit, SW\_SHOW);

ShowWindow(hBtnOk, SW\_SHOW);

ShowWindow(hComboBox, SW\_SHOW);

action = ACTION\_SET\_VALUE;

break;

}

case ID\_DELETE\_VALUE: {

DeleteRegistryValue(hwnd, valueName);

FillDataAboutKey(0);

ShowRegistryKeysAndValues(hwnd);

break;

}

case ID\_CREATE\_KEY: {

ShowWindow(hEdit, SW\_SHOW);

ShowWindow(hBtnOk, SW\_SHOW);

action = ACTION\_CREATE\_KEY;

break;

}

case ID\_DELETE\_KEY: {

DeleteRegistryKey(hwnd, chosenFile[pointer]);

break;

}

case ID\_SET\_DATA: {

switch (action) {

case ACTION\_CREATE\_KEY: {

int length = GetWindowTextA(hEdit, valueName, sizeof(valueName));

action = 0;

ShowWindow(hEdit, SW\_HIDE);

ShowWindow(hBtnOk, SW\_HIDE);

SetWindowTextA(hEdit, "");

CreateRegistryKey(hwnd, valueName);

break;

}

case ACTION\_SET\_VALUE: {

int length = GetWindowTextA(hEdit, newValue, sizeof(newValue));

char tmp[MAX\_KEY\_LENGTH] = { '\0' };

GetWindowTextA(hComboBox, tmp, sizeof(tmp));

action = 0;

ShowWindow(hEdit, SW\_HIDE);

SetWindowTextA(hEdit, "");

ShowWindow(hBtnOk, SW\_HIDE);

ShowWindow(hComboBox, SW\_HIDE);

SetRegistryValue(hwnd, valueName, newValue, GetRegType(tmp));

break;

}

case ACTION\_CREATE\_VALUE: {

int length = GetWindowTextA(hEdit, valueName, sizeof(valueName));

action = 0;

ShowWindow(hEdit, SW\_HIDE);

ShowWindow(hBtnOk, SW\_HIDE);

SetWindowTextA(hEdit, "");

SetRegistryValue(hwnd, valueName, "", REG\_SZ);

FillDataAboutKey(0);

ShowRegistryKeysAndValues(hwnd);

break;

}

}

break;

}

case ID\_BACK: {

pointer--;

if (pointer > 0) {

OpenRegistryKey(hwnd, chosenFile[pointer], 0);

chosenFile[pointer + 1][0] = '\0';

RegCloseKey(hKeys[pointer + 1]);

hKeys[pointer + 1] = NULL;

}

else if (pointer == 0)

OpenRegistryKeyForFirst(hwnd);

else pointer++;

ShowRegistryKeysAndValues(hwnd);

break;

}

case ID\_CREATE\_VALUE: {

ShowWindow(hEdit, SW\_SHOW);

ShowWindow(hBtnOk, SW\_SHOW);

action = ACTION\_CREATE\_VALUE;

break;

}

case ID\_READ\_TYPE: {

ReadRegistryType(hwnd, valueName);

break;

}

}

if (HIWORD(wParam) == LBN\_SELCHANGE) {

if ((HWND)lParam == hListBoxKeys) {

int index = (int)SendMessage(hListBoxKeys, LB\_GETCURSEL, 0, 0);

int length = (int)SendMessage(hListBoxKeys, LB\_GETTEXTLEN, index, 0);

length++;

SendMessageA(hListBoxKeys, LB\_GETTEXT, index, (LPARAM)chosenFile[pointer]);

}

}

if (HIWORD(wParam) == LBN\_SELCHANGE && (HWND)lParam == hListBoxValues) {

int index = (int)SendMessage(hListBoxValues, LB\_GETCURSEL, 0, 0);

int length = (int)SendMessage(hListBoxValues, LB\_GETTEXTLEN, index, 0);

length++;

SendMessageA(hListBoxValues, LB\_GETTEXT, index, (LPARAM)valueName);

}

break;

}

case WM\_DESTROY: {

for (int i = 0; i <= pointer; i++)

RegCloseKey(hKeys[i]);

PostQuitMessage(0);

break;

}

default:

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

}

return 0;

}

int OpenRegistryKey(HWND hwnd, const char\* keyName, int i) {

if (RegOpenKeyExA(hKeys[pointer], keyName, 0, KEY\_ALL\_ACCESS, &hKeys[pointer + 1]) == ERROR\_SUCCESS && (keyName != NULL && keyName[0] != '\0')) {

FillDataAboutKey(i);

ShowRegistryKeysAndValues(hwnd);

return 0;

}

else {

MessageBoxA(hwnd, "Failed to open key.", "Error", MB\_OK | MB\_ICONERROR);

return 1;

}

}

void ReadRegistryValue(HWND hwnd, const char\* valueName) {

LPDWORD dataType = REG\_SZ;

LPBYTE data[256] = {'\0'};

LPDWORD dataSize = sizeof(data);

if (RegQueryValueExA(hKeys[pointer], valueName, NULL, &dataType, data, &dataSize) == ERROR\_SUCCESS)

MessageBoxA(hwnd, (unsigned short\*)data, "Value", MB\_OK);

else

MessageBoxA(hwnd, "Failed to read value.", "Error", MB\_OK | MB\_ICONERROR);

}

void SetRegistryValue(HWND hwnd, const char\* valueName, const char\* newValue, DWORD type){

if (RegSetValueExA(hKeys[pointer], valueName, 0, type, (const BYTE\*)newValue, strlen(newValue) + 1) != ERROR\_SUCCESS)

MessageBoxA(hwnd, "Failed to set value.", "Error", MB\_OK | MB\_ICONERROR);

else

MessageBoxA(hwnd, "Value set successfully.", "Info", MB\_OK);

}

void DeleteRegistryValue(HWND hwnd, const char\* valueName) {

if (RegDeleteValueA(hKeys[pointer], valueName) != ERROR\_SUCCESS)

MessageBoxA(hwnd, "Failed to delete value.", "Error", MB\_OK | MB\_ICONERROR);

else

MessageBoxA(hwnd, "Value deleted successfully.", "Info", MB\_OK);

}

void CreateRegistryKey(HWND hwnd, const char\* valueName) {

HKEY hk;

DWORD disposition;

if (RegCreateKeyExA(hKeys[pointer], valueName, 0, NULL, REG\_OPTION\_NON\_VOLATILE, KEY\_WRITE, NULL, &hk, &disposition) == ERROR\_SUCCESS) {

FillDataAboutKey(0);

ShowRegistryKeysAndValues(hwnd);

}

else {

MessageBoxA(hwnd, "Failed to create key.", "Error", MB\_OK | MB\_ICONERROR);

}

}

void DeleteRegistryKey(HWND hwnd, const char\* valueName) {

if (RegDeleteKeyExA(hKeys[pointer], valueName, KEY\_WOW64\_64KEY, 0) == ERROR\_SUCCESS) {

FillDataAboutKey(0);

ShowRegistryKeysAndValues(hwnd);

}

else {

MessageBoxA(hwnd, "Failed to delete key.", "Error", MB\_OK | MB\_ICONERROR);

}

}

void ShowRegistryKeysAndValues(HWND hwnd) {

int heigt = 150 / 9 \* subKeyCount;

SendMessageA(hListBoxKeys, LB\_RESETCONTENT, 0, 0);

SetWindowPos(hListBoxKeys, NULL, 10, 70, 300, heigt, SWP\_NOZORDER);

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

SendMessageA(hListBoxKeys, LB\_ADDSTRING, 0, (LPARAM)keys[i].name);

}

heigt = 150 / 9 \* valueCount;

SendMessageA(hListBoxValues, LB\_RESETCONTENT, 0, 0);

SetWindowPos(hListBoxValues, NULL, 320, 70, 200, heigt, SWP\_NOZORDER);

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

SendMessageA(hListBoxValues, LB\_ADDSTRING, 0, (LPARAM)values[i].name);

}

}

void FillDataAboutKey(int i) {

DWORD valueNameSize, valueDataSize;

RegQueryInfoKey(hKeys[pointer + i], NULL, NULL, NULL, &subKeyCount, NULL, NULL, &valueCount, NULL, NULL, NULL, NULL);

for (DWORD index = 0; index < subKeyCount && index < MAX\_KEYS; ++index) {

DWORD keyNameSize = MAX\_KEY\_LENGTH;

RegEnumKeyExA(hKeys[pointer + i], index, keys[index].name, &keyNameSize, NULL, NULL, NULL, NULL);

}

for (DWORD index = 0; index < valueCount && index < MAX\_VALUES; ++index) {

valueNameSize = MAX\_VALUE\_NAME;

valueDataSize = MAX\_VALUE\_NAME;

values[index].dataType = 0;

RegEnumValueA(hKeys[pointer + i], index, values[index].name, &valueNameSize, NULL, &values[index].dataType, values[index].data, &valueDataSize) == ERROR\_SUCCESS;

}

}

int OpenRegistryKeyForFirst(HWND hwnd) {

if (RegOpenKeyExA(HKEY\_CURRENT\_USER, NULL, 0, KEY\_ALL\_ACCESS, &hKeys[pointer]) == ERROR\_SUCCESS) {

FillDataAboutKey(0);

ShowRegistryKeysAndValues(hwnd);

return 0;

}

else {

MessageBoxA(hwnd, "Failed to open key.", "Error", MB\_OK | MB\_ICONERROR);

return 1;

}

}

void ReadRegistryType(HWND hwnd, const char\* valueName) {

LPDWORD dataType = REG\_SZ;

LPBYTE data[256] = { '\0' };

LPDWORD dataSize = sizeof(data);

if (RegQueryValueExA(hKeys[pointer], valueName, NULL, &dataType, data, &dataSize) == ERROR\_SUCCESS)

MessageBoxA(hwnd, (unsigned short\*)GetRegTypeName(dataType), "Value", MB\_OK);

else

MessageBoxA(hwnd, "Failed to read value.", "Error", MB\_OK | MB\_ICONERROR);

};

const char\* GetRegTypeName(DWORD dataType) {

switch (dataType) {

case REG\_SZ:

return "REG\_SZ";

case REG\_DWORD:

return "REG\_DWORD";

case REG\_BINARY:

return "REG\_BINARY";

case REG\_EXPAND\_SZ:

return "REG\_EXPAND\_SZ";

case REG\_MULTI\_SZ:

return "REG\_MULTI\_SZ";

default:

return "Unknown Type";

}

}

DWORD GetRegType(const char\* type) {

if (strcmp(type, "REG\_SZ") == 0)

return REG\_SZ;

else if (strcmp(type, "REG\_DWORD") == 0)

return REG\_DWORD;

else if (strcmp(type, "REG\_BINARY") == 0)

return REG\_BINARY;

else if (strcmp(type, "REG\_EXPAND\_SZ") == 0)

return REG\_EXPAND\_SZ;

else if (strcmp(type, "REG\_MULTI\_SZ") == 0)

return REG\_MULTI\_SZ;

}

int GetIdOfType(DWORD type) {

char\* items[] = { "REG\_SZ", "REG\_DWORD", "REG\_BINARY", "REG\_EXPAND\_SZ", "REG\_MULTI\_SZ", NULL };

for (int i = 0; items[i] != NULL; i++) {

if (strcmp(items[i], GetRegTypeName(type)) == 0)

return i;

}

return -1;

}

DWORD GetType(HWND hwnd, const char\* valueName) {

LPDWORD dataType = REG\_SZ;

LPBYTE data[256] = { '\0' };

LPDWORD dataSize = sizeof(data);

if (RegQueryValueExA(hKeys[pointer], valueName, NULL, &dataType, data, &dataSize) == ERROR\_SUCCESS)

return dataType;

return REG\_SZ;

}

char\* GetData(HWND hwnd, const char\* valueName) {

LPDWORD dataType = REG\_SZ;

LPBYTE data[256] = { '\0' };

LPDWORD dataSize = sizeof(data);

if (RegQueryValueExA(hKeys[pointer], valueName, NULL, &dataType, data, &dataSize) == ERROR\_SUCCESS)

return data;

return '\0';

}