Win32 api 프로그래밍 구조

메인 함수가 2가지로 나뉨
1) WinMain
-윈도우 구조체 등록, 생성
-윈도우 생성과 출력
-메세지 루프, 메세지 체크와 전달 (프로그램을 종료하라는 메세지를 받을 때 까지 대기하는 부분)

```
while( GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg); // WinProcedure에 메세지 전달 후 리턴받을 때까지 대기
}
```

-windows 프로그램의 진입점

2) WinProcedure
-메세지가 발생했을 때 처리
(WM_xxx, CB_xxx, LM_xxx, PBM_xxx 등의 메세지 형태가 존재한다)
-switch(message) 와 같은 형태로 메세지를 처리한다
default 로는 DefWndProc()의 함수로 처리하게 한다 (이미 처리방법이 함수에 정의되어 있음)

문자 집합
1) 멀티 바이트 문자 집합(MBCS)
- 1 ~ 2바이트 크키의 문자집합
-영문과 한글 지원, 일반 C 함수는 MBCS 문자집합 사용

2) 유니코드 문자 집합
-2 바이트로 전세계 언어를 표현하는 문자 집합
-유니 코드는 멀티 바이트에 비해 메모리는 많이 차지하지만 처리속도는 더 빠르다.
-아스키 코드 값의 일부는 유니코드와 값이 같다

LPTSTR str = TEXT("문자열")
=>TEXT 는 ANSI 문자열을 유니코드로 변환 매크로 함수이다

WndProc() 기본 메세지

1) WM_PAINT
-최초 UpdateWindow() 함수에 의해 발생
-윈도우의 일부 영역을 새로 출력할 때 발생

2) WM_DESTORY
-윈도우가 메모리에서 제거되기 (WM_QUIT) 직전에 발생
- 이 함수가 호출될 때는 윈도우 화면이 사라졌지만, 메모리에는 있는 상태임

리소스
-커서, 메뉴, 아이콘, 문자열, 비트맵, 다이얼로그, 엑셀레이터
-스크립터 언어로 작성됨
-리소스는 아이디(중복되지 않는 양의 정수 값)로 다룬다
-resource.h 에 정의되어 있다 => #define

아이콘
-왼쪽 상단과 최소화 할 때 하단 제목 표시줄에 나타나는 작은 이미지
-*.ico 파일 확장자
-아이콘 리소스 접두사 : IDI_xxx 형태

아이콘 로딩 함수
```
HICON LoadIcon(
        __in HINSTANCE hInstance,
        __in LPCTSTR lpIconName // 아이콘 아이디 정수값을 문자열 형식으로 지정
);
```

아이콘 아이디 정수값을 문자열 형식으로 지정 매크로 함수 사용
```
LPTSTR MAKEINTRESOURCE(
        WORD wInteger
)
```

커서 리소스
-커서 설정은 아이콘과 유사하다
-접두사 : 'IDC_', *.cur
-운영체제에서 제공하는 기본 커서 아이디가 존재한다

-로딩 함수(기본 커서 로딩 및 적용)
```
HCURSOR LoadCursor(
        __in HINSTANCE hInstance, // NULL
        __in LPCTSTR lpCursorName
);
```

-핫 스폿(hot spot) : 커서가 윈도우 영역 안에 있는지 없는지를 판단하는 좌표

WM_SETCURSOR를 이용한 커서 설정
-마우스 이동할 때 발생하는 메세지
-WM_MOUSEMOVE, WM_SETCURSOR
-SetCursor()로 커서 설정
```
-HCURSOR SetCursor(
        __int HCURSOR hCursor
);
```

메뉴 리소스
-리소스 편집기의 Menu->메뉴삽입->에서 그냥 만들면 됨
-메뉴 목록을 만들면 visual studio 에서 자체적으로 ID를 발급해줌 (구글링 해보면 탭이 나옴)
-메뉴 항목을 누르면 WM_COMMAND 메세지 발생한다
-각각의 메뉴 ID는 WinProc의 LOWORD(wParam)에 값이 들어온다

```
int WINAPI MessageBox(
        _In_opt_HWND hWnd, // 윈도우 핸들
        _In_opt_LPCTSTR lpText, // 내가 출력하고자 하는 문자열
        _In_opt_LPCTSTR lpCaption, // 타이틀 바
        _In_UINT uType, // 메세지 바
);
```

그래픽
-클라이언트 영역에 원하는 것들을 출력할 수 있도록 하는 것
-운영체제의 한 부분인 GDI는 출력을 담당한다 (Gdi.dll 에 존재)
-GDI는 아래에 있는 Graphics drivers 를 통제한다
-DC는 출력하기 위한 장치(화면, 프린터 ..)에 대한 특성을 저장하는 구조체
즉 구조가, Application -> DC(화면, 프린터) -> GDI 의 구조를 가지고 있다

-DC를 사용하는 그래픽 오브젝트는 비트맵, 브러쉬, 펜, 팔레트, 폰트 등이 있다
-DC의 데이터형 : 핸들
-화면 DC에 관련된 함수
(ex. BeginPaint(), EndPaint(), GetDC(), ReleaseDC()) => 사용했으면 OS에 반환하는 과정을 거친다

1) WM_PAINT 메세지 유형
-HDC BeginPaint(HWND hwnd, LPPAINTSTRUCT lpPaint) => DC를 가져오는 역할
-BOOL EndPaint(HWND hwnd, const PAINTSTRUCT *lpPaint)

2)
-HDC GetDC(HWND hWnd);
-int ReleaseDC(HWND hWnd, HDC hDC)

문자 출력

-BOOL TextOut(HDC hdc, int nXStart, int nYStart, LPCTSTR lpString, int cbString)
// 1번째 매개변수가 DC임
// 2, 3번째 매개변수는 출력할 좌표
// lpString : 문자열 (윈도우에서 출력하는 모든 것은 문자열로 바꿔야 한다) => sprintf
// cbString : 출력하고자 하는 문자열의 길이 => strlen

InvalidateRect() : 화면의 일부 또는 전체를 다시 출력할 때
(화면의 일부를 지정할 때는 RECT 구조체를 사용한다,) => WM_PAINT 메세지를 발생시킨다
-업그레이드 영역(다시 그리고자 하는 영역을 지정한다)

BOOL **InvalidateRect**(HWND hWnd, **const RECT *lpRect**, BOOL bErase);
typedef struct _RECT { LONG left, LONG top; LONG right; LONG bottom; }RECT *PRECT;

그래픽 오브젝트(GDI 오브젝트)
-PEN, Brush, Memory DC, Font ...

GDI 오브젝트 사용 방법
-운영체제에서 제공, 사용자가 직접 설정

스톡오브젝트 : OS 에서 제공하는 GDI 오브젝트를 사용하기 위한 함수
1) GetStockObject() : GDI 오브젝트에 대한 핸들을 얻을 수 있다

원형 : HGDIOBJ GetStockObject(int fnObject)
브러쉬 : BLACK_BRUSH, DKGRAY_BRUSH, WHITE_BRUSH ..
펜 : BLACK_PEN, DC_PEN...
폰트 : ANSI_FIXED_FONT, ...

2) SelectObject() : GDI 오브젝트 설정

원형 : HGDIOBJ SelectObject(HDC hdc, HGDIOBJ hgdiobj)

3) DeleteObject() : GDI 오브젝트 해제
// 스톡 오브젝트 같은 경우는 DeleteObject 를 할 필요가 없다. 직접 만들어서써야 위 과정 해야함

펜
-선 색상, 선 굵기, 스타일 지정(점선, 실선등)
HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF crColor);

펜을 이용한 선 그리기
BOOL MoveToEx(HDC hdc, int x, int y, LPPOINT lpPoint) => 그리는 시작 좌표 설정
LineTo (HDC hdc, int nxEnd, int nyEnd); => 종료되는 좌표 설정

정리
1) 펜 생성 : CreatePen() => 사용자가 생성할 경우, GetStockObject() => 운영체제꺼 사용할 경우
2) 펜 선택 : SelectObject()
3) 선 그리기 : MoveToEx() , LineTo()
4) 펜 제거 및 이전 펜 설정 : DeleteObject(), SelectObject() => 스톡 오브젝트는 제외

비트맵

-이미지의 종류(bmp, jpg, gif, tga) 에 대해 이미지를 불러들이는 방법이 다르다

비트맵을 다루는 두 가지 방법

1) 비트맵을 리소스에 등록하여 사용 (visual studio .rc 파일 – 리소스 추가 – Bitamp – 가져오기)

2) LoadImage() 함수를 이용하여 파일로부터 읽어내는 방법

비트맵을 읽어 출력하는 순서

1) 비트맵의 핸들을 얻는다 : LoadBitmap(), LoadImage()

HBITMAP LoadBitmap(HINSTANCE hInstance, LPCTSTR lpBitmapName);

HANDLE LoadImage(HINSTANCE hinst, LPCTSTR lpszName, UINT uType, int cxDesired, int cyDesired...)

2) 메모리 DC 생성 : CreateCompatibleDC() => 이미지를 메모리에 적재

HDC CreateCompatibleDC(HDC hdc)

3) 메모리 DC에 비트맵 적용 : SelectObject()

4) 비트맵 출력 : BitBIt()

BOOL BitBLt (HDC hdcDest, int nxDest, int nyDest, int nWidth, int nHeight, HDC hdcSrc, int nxSrc, int nySrc, DWORD dwRop)

5) 메모리 DC와 비트맵 제거 : DeleteDC(), DeleteObject()

키보드 메세지

1) **WM_CHAR** : 문자 키에 발생하는 메세지인데, 어떤 문자가 눌렸는지는 알 수 없음

-WinProc의 wParam에 아스키 코드 값으로 넘어옴

InvalidateRect() : 화면의 일부 또는 전체를 다시 출력할 때

(화면의 일부를 지정할 때는 RECT 구조체를 사용한다,) => WM_PAINT 메세지를 발생시킨다

그 후에 WM_PAINT 에서 코드 작성

2) **WM_KEYDOWN** : 모든 키에 대해 발생하는 메세지

-WM_CHAR 과 같이 wParam에 아스키 코드값이 들어오는게 아니고 가상 키 코드가 들어온다. 문자가 눌려지면 모두 대문자로 인식한다.

ex) switch(wParam)

```
        case VK_LEFT:
                MessageBox( hwnd, "Left key" ... )
```

GetAsyncKeyState()

-실시간으로 키 입력을 체크

-순서대로 처리해야 되는 메세지 큐의 단점을 보완

-키 눌림이 있으면 음수값 리턴

-원형 : SHORT GetAsyncKeyState(int nKey)

ex)

if(GetAsyncKeyState(VK_LEFT) < 0) {} 와 같이 사용한다

마우스 메세지

1) **WM_MOUSEMOVE** : 마우스 이동시 발생

마우스 위치정보

-LOWORD(lParam) => x 좌표

-HIWORD(lParam) => y 좌표

ex)

```
case WM_MOUSEMOVE:
        nx = LOWORD( lparam);
        ny = LOWORD( lparam);
        InvalidateRect(hwnd, NULL, ...);
case WM_PAINT
        .....
        현재 마우스의 x, y 값을 TextOut 으로 출력한다
```

2) WM_LBUTTONDOWN, WM_LBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP ..

드래그 : WM_MOUSEMOVE + MK_LBUTTON(마우스 왼쪽 버튼이 눌려져 있음)

ex)

```
case WM_MOUSEMOVE:
        if( wParam == MK_LBUTTON) {
                sprintf(string, "%s", "마우스 드래그");
        }
        else {
                sprintf( string, "%s", "마우스 이동중");
        }
        InvalidateRect( hWnd, NULL, ...)
```

타이머

-일정한 시간 간격으로 함수 호출 또는 WM_TIMER 메세지 발생

-사용 용도 : 일정한 시간 간격으로 코드 실행

-생성

```
UINT_PTR SetTimer(
        HWND hWnd,
        UINT_PTR nIDEvent, // 타이머의 중복되지 않는 ID로 세팅
        UINT uElapse,   // 함수 호출의 시간 간격
        TIMERPROC lpTimerFunc // 호출할 함수명 지정
)
```

-해제

```
BOOL KillTimer(
        HWND hWnd,
        UINT_PTR uIDEvent)
```

+UINT_PTR : unsigned int, unsigned_int64

타이머 이벤트 처리

-메세지 처리 : WM_TIMER + wParam(타이머 아이디)

ex)

```
case WM_CREATE:   // CreateWindow할 때 WM_CREATE 메세지가 생성
        SetTimer( hWnd, 1, 1000, NULL);
        SetTimer (hWnd, 2, 10000, NULL);
case WM_TIMER:
        switch(wParam) {
                case 1:
                case 2:
        }
```

-콜백 함수 처리

```
VOID CALLBACK 함수이름 (
        HWND hwnd,
        UINT uMsg,
        UINT_PTR idEvent,
        DWORD dwTime
)
```

다이얼로그

1) 모달형

-다이얼로그가 최우선인 순위

-대표적인 다이얼로그 형태는 MessageBox() 이다

다이얼로그 생성

```
INT_PTR DialogBox(
        HINSTANCE hInstance,
        LPCTSTR lpTemplate,
        HWND hWndParent,
        DLGPROC lpDialogFunc
)
```

DialogProc와 WndProc의 차이점

-WndProc

메세지 처리 => DefWindowProc

-DialogProc

메세지 처리 => TRUE, FALSE

2) 모델리스형

모델리스형 생성

```
HWND CreateDialog(
        HINSTANCE hInstance,
        LPCTSTR lpTemplate,
        HWND hWndParent,
        DLGPROC lpDialogFunc
)
```

모델리스형 다이얼로그 해제

```
BOOL DestroyWindow( HWND hWnd)
```

=>WM_CLOSE 에서 호출

```
// Win32Project3.cpp : 응용 프로그램에 대한 진입점을 정의합니다.
//

#include "stdafx.h"
#include "Win32Project4.h"
#include <stdio.h>

#define MAX_LOADSTRING 100

// 전역 변수:
HWND g_hWnd;
HINSTANCE hInst;                                // 현재 인스턴스입니다.
WCHAR szTitle[MAX_LOADSTRING];                  // 제목 표시줄 텍스트입니다. (visual studio 기준으로 위에 Win32Project3 - Microsoft Visual Studio 부분)
WCHAR szWindowClass[MAX_LOADSTRING];            // 기본 창 클래스 이름입니다. 구조체 이름 ( 이전에 basic 과 같은 문자열을 배열으로 저장)

                                                                                // 이 코드 모듈에 들어 있는 함수의 정방향 선언입니다.
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); // 버전에 따라 붙을 수도 있고 없을 수도 있다
        UNREFERENCED_PARAMETER(lpCmdLine); // 버전에 따라 붙을 수도 있고 없을 수도 있다

                                                                        // TODO: 여기에 코드를 입력합니다.

                                                                        // 전역 문자열을 초기화합니다.
        LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING); // Ids_app_title : .rsc 에 들어가서 string table 에서 상단의 Win32Project3- Microsoft visual studio 제목 변경 가능
        LoadStringW(hInstance, IDC_WIN32PROJECT4, szWindowClass, MAX_LOADSTRING);

        // 문자열을 리소스로 다룰 수 있는데 문자열에 ID를 부여하고 ID를 활용해서 배열에 복사할 수 있다
        MyRegisterClass(hInstance);
```

```
	// 응용 프로그램 초기화를 수행합니다.
	if (!InitInstance(hInstance, nCmdShow)) // 윈도우 생성
	{
		return FALSE;
	}

	HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_WIN32PROJECT4));
	// hAccelTable : 단축키 부분임

	MSG msg;

	// 기본 메시지 루프입니다.
	while (GetMessage(&msg, nullptr, 0, 0))
		// 시스템의 메세지 큐에서 메세지를 읽어 들인다, 메세지는 &msg에 저장된다
		// 읽어들인 메세지가 시스템을 종료하라는 WM_QUIT 이면 FALSE를 리턴하고 그 외에는
TRUE를 리턴한다
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

		{
			TranslateMessage(&msg);
			// 이 함수는 키보드의 눌림(WM_KEYDOWN)과 떨어짐(WM_KEYUP)이 연속적으로
발생할 때 문자가 입력되었다는 메시지(WM_CHAR)를 만드는 역할을 한다
			DispatchMessage(&msg);
			//시스템 메시지 큐에서 꺼낸 메시지를 프로그램의 메시지 처리 함수(WndProc)로
전달한다.
		}
	}

	return (int)msg.wParam;
}



//
//  함수: MyRegisterClass()
//
//  목적: 창 클래스를 등록합니다.
//
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_WIN32PROJECT4)); // 다른 아이콘
```

```
	wcex.hCursor = LoadCursor(nullptr, IDC_ARROW); // 커서 지정 NULL, IDC_ARROW : 운영체제 기본 커서 아이디 사용
	// hCursor에서 운영체제에서 기본적으로 제공하는 것을 사용하지 않고 임의의 커서를 적용할 경우 hInstance, MAKEINTRESOURCE(IDC..) 이렇게 작성

	wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
	wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_WIN32PROJECT4); // 메뉴 리소스 만들 때
	wcex.lpszClassName = szWindowClass;
	wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL)); // small 아이콘

	return RegisterClassExW(&wcex); // 구조체 등록
}

//
//   함수: InitInstance(HINSTANCE, int)
//
//   목적: 인스턴스 핸들을 저장하고 주 창을 만듭니다.
//
//   설명:
//
//        이 함수를 통해 인스턴스 핸들을 전역 변수에 저장하고
//        주 프로그램 창을 만든 다음 표시합니다.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) // 등록된 윈도우로 윈도우 생성
{
	hInst = hInstance; // 인스턴스 핸들을 전역 변수에 저장합니다.

	HWND hWnd;
	// CreateWindowW : 윈도우 생성
	g_hWnd = hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
		CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);


	if (!hWnd)
	{
		return FALSE;
	}

	// nCmdShow : WinMain의 매개변수
	ShowWindow(hWnd, nCmdShow); // 메모리에 있는 윈도우를 스크린상에 출력한다
	UpdateWindow(hWnd); // 클라이언트 영역을 새로 그린다

	return TRUE;
}

//
//  함수: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  목적:  주 창의 메시지를 처리합니다.
//
//  WM_COMMAND  - 응용 프로그램 메뉴를 처리합니다.
```

```
//  WM_PAINT    - 주 창을 그립니다.
//  WM_DESTROY  - 종료 메시지를 게시하고 반환합니다.
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

        HCURSOR hCursor;

        switch (message)
        {
        //case WM_SETCURSOR:
        //      hCursor = LoadCursor(hInst, MAKEINTRESOURCE(IDC_CURSOR...))
        //      break;  => 자체적으로 커서를 설정하는경우 위와 같이 작성한다

        case WM_COMMAND:
        {
                int wmId = LOWORD(wParam);

                // 메뉴 선택을 구문 분석합니다.
                switch (wmId)
                {

                //case ID_32771: // 작성한 메뉴1 => 메뉴를 새로 작성하면 해당 메뉴ID를 작성한다
                //      MessageBox(0, "열기", "연습", MB_OK);
                //      break;

                case IDM_ABOUT:
                        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                        break;
                case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
        }
        break;
        case WM_PAINT: // 화면 출력
        {
                char string[100];
                PAINTSTRUCT ps; // 화면을 그리기 위한 모든 셋업값을 구조체에 저장한다
                HDC hdc = BeginPaint(hWnd, &ps); // 화면 그리기 준비, hdc : 화면 핸들
                                                                                // TODO: 여기에 hdc를
사용하는 그리기 코드를 추가합니다.

                //sprintf(string, "%d %f %s", 12, 3.14f, "win32");
                //TextOut(hdc, 0, 0, string, strlen(string));

                EndPaint(hWnd, &ps); // 화면 그리기 종료
        }
```

```
        break;
        case WM_DESTROY:
                PostQuitMessage(0); // WM_QUIT 발생
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}

// 정보 대화 상자의 메시지 처리기입니다.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

// 위 프로그램을 실행하면 도움말-정보(A)... 가 뜨는데 이부분 처리를 담당
{
        UNREFERENCED_PARAMETER(lParam);
        switch (message)
        {
        case WM_INITDIALOG:
                return (INT_PTR)TRUE;

        case WM_COMMAND:
                if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                {
                        EndDialog(hDlg, LOWORD(wParam));
                        return (INT_PTR)TRUE;
                }
                break;
        }
        return (INT_PTR)FALSE;
}
```