[목 차]

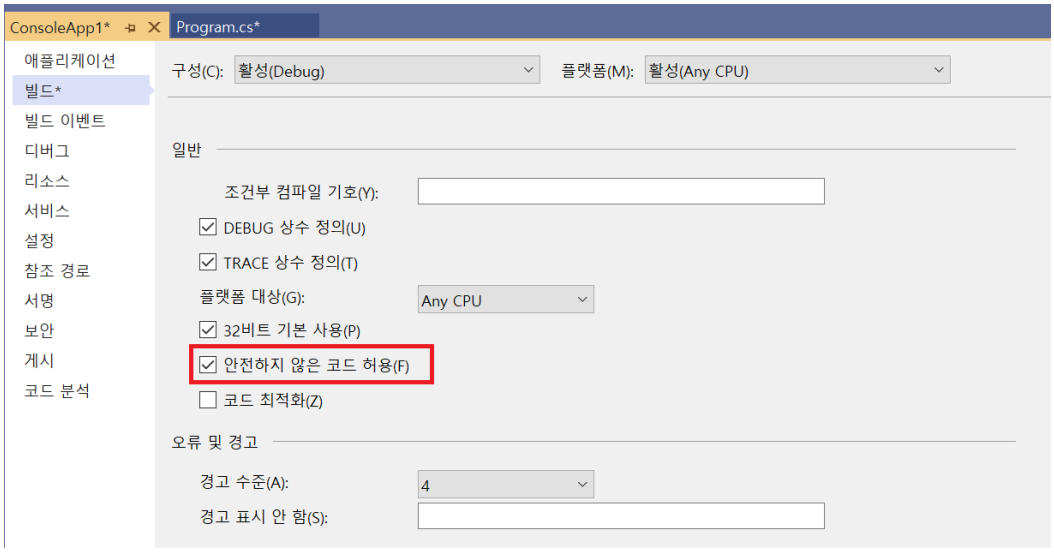
1. 안전하지 않은 컨텍스트(unsafe)

기존 네이티브 C/C++ 언어와의 호환성을 위한 기능이다.

안전하지 않은 컨텍스트란 안전하지 않은 코드를 포함한 영역을 의미하며, 안전하지 않은 코드란 포인터 사용하는 것을 의미한다.

C#은 C/C++언어의 포인터를 지원하며 unsafe 예약어는 포인터를 쓰는 코드를 포함하는 클래스나 그것의 맴버 또는 블록에 사용한다.

속성창에서 "안전하지 않은 코드 허용" 옵션을 설정해야 한다.




```
using System;
namespace Sample{
    class Program    {
        unsafe static void GetAddResult(int* p, int a, int b)
        {
            *p = a + b;
        }
        static void Main(string[] args)          {
            int i;
            unsafe                {
                GetAddResult(&i, 5, 10);
            }
            Console.WriteLine(i);
        }
    }
}
```

2. 참조 형식의 맴버에 대한 포인터(fixed)

unsafe 문맥에서 포인터를 사용할 수 있는 것은 스택에 데이터가 저장된 변수에 한해 적용된다. 즉, 지역 변수나 메서드의 매개변수 타입이 값 형식인 경우에만 포인터 연산자(&, *)를 사용할 수 있다.

반면 참조 형식의 데이터는 직접적인 포인터 연산을 지원할 수 없다. 이유는 참조 형식의 인스턴스는 힙에 할당되고, 그 데이터는 가비지 수집기가 동작할 때마다 위치가 바뀔 수 있기 때문이다.

Fixed는 힙에 할당된 참조 형식의 인스턴스를 가비지 수집기가 움직이지 못하도록 고정시킴으로써 포인터가 가리키는 메모리를 유효하게 만든다.

보통 fixed된 포인터는 관리 프로그램의 힙에 할당된 데이터를 관리되지 않은 프로그램에 넘기는 용도로 쓰인다.

```
using System;
namespace Sample {
    class Managed    {        public int Count;        public string Name;    }
    class Program    {
        //객체 인스턴스 포인터를 가져오는 것을 허용하지 않는다.
        //해당 맴버가 가진 맴버 데이터가 값 형식이거나 값 형식 배열인 경우에만 가능.
        unsafe static void Main(string[] args)        {
            Managed inst = new Managed()            {
                Count = 5,
                Name = "test"
            };
            fixed (int* pValue = &inst.Count)                {
                *pValue = 6;
                Console.WriteLine(inst.Count);   //6
            }
            fixed (char* pChar = inst.Name.ToCharArray())            {
                for (int i = 0; i < inst.Name.Length; i++)
                {
                    Console.Write(*(pChar + i));       //test
                }
                Console.WriteLine();
            }
        }
    }
}
```

3. 스택을 이용한 값 형식 배열(stackalloc)

값 형식은 스택에 할당되고 참조 형식은 힙에 할당된다. 그런데 값 형식 임에도 그것이 배열로 선언되면 힙에 할당된다.

Stackalloc 예약어는 값 형식의 배열을 힙이 아닌 스택에 할당한다.

포인터 연산을 사용하기 때문에 stackalloc 도 unsafe 문에서 사용해야 한다.

왜 스택에 배열을 만들까? 이유는 힙을 사용하지 않으므로 가비지 수집기의 부하가 없다는 장점이 있다. 끊임없이 호출되는 메서드 내에서 힙에 메모리를 할당하면 가비지 수집기로 인해 끊김 현상이 발생할 수 있다. 이럴 때 stackalloc 을 이용하면 가비지 수집기의 호출 빈도를 조금이라도 낮출 수 있다.

반대로 왜 스택에 배열을 만들고 싶지 않을까? 스택은 1M 정도의 제한된 자원을 사용하기 때문에 신중을 기해야 한다.

```
using System;

namespace Sample
{
    class Program
    {
        unsafe static void Main(string[] args)
        {
            int* pArray = stackalloc int[1024]; // int 4byte * 1024 == 4KB 용량을 스택에 할당

            for(int i=0; i< 1024; i++)
            {
                pArray[i] = i;
                Console.Write(pArray[i] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

4. IntPtr(정수형 포인터)

IntPtr은 정수형 포인터를 의미하는 값 형식의 타입이다.

[사용1] 메모리 주소 보관

포인터는 메모리 주소를 보관하는 곳이므로 32비트 프로그램에서는 2^{32} 주소 영역을 지정할 수 있어야 하고, 64비트 프로그램에서는 2^{64} 주소 영역을 지정할 수 있어야 한다.

이 때문에 IntPtr 자료형은 32비트 프로그램에서는 4byte 64비트 프로그램에서는 8byte로 동작한다.

[사용2] 윈도우 운영체제의 핸들(HANDLE) 보관

핸들은 윈도우 운영체제가 특정 자원에 대한 식별자 이다.

```
using System;
using System.IO;

namespace Sample
{
    class Program
    {
        [Obsolete]
        static void Main(string[] args)
        {
            Console.WriteLine(IntPtr.Size);
            // 출력결과
            // 32비트 프로그램인 경우: 4
            // 64비트 프로그램인 경우: 8

            using (FileStream fs = new FileStream("test.dat", FileMode.Create))
            {
                //public virtual IntPtr Handle { get; }
                Console.WriteLine(fs.Handle);
                Console.WriteLine(fs.SafeFileHandle);
            }
        }

    }
}
```

5. Win32 API 호출(extern)

닷넷 호환 언어로 만들어진 관리 코드에서 C/C++와 같은 언어로 만들어진 비관리 코드의 기능을 사용하는 수단으로 플랫폼 호출(P/Invoke: Platform invocation)이 있다. extern 예약어는 C#에서 PInvoke 호출을 정의하는 구문에 사용된다.

extern 구문을 작성하려면 3가지 정보가 필요하다.

- 비관리 코드를 제공하는 DLL 이름
- 비관리 코드 함수 명
- 비관리 코드의 함수 형식

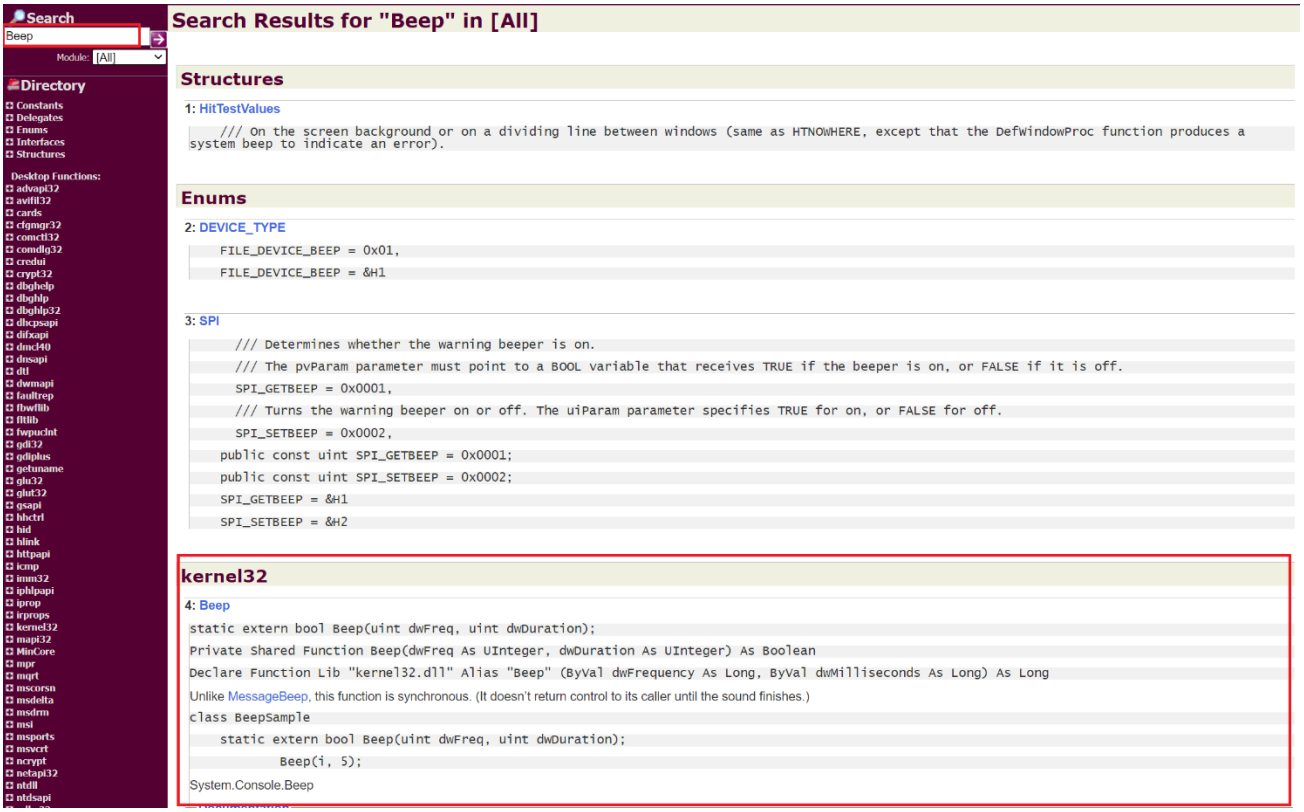
C#의 P/Invoke 방식은 런타임시에 동적으로 함수를 호출하는 방식이다. 따라서 컴파일시 에러를 체크하지 못한다.

Win32 에 있는 함수들을 C# 메서드 원형으로 변환하기 위해서는 pinvoke.net 사이트를 이용하면 편리하다.

사이트 : pinvoke.net: the interop wiki!

Pinvoke 툴:

https://marketplace.visualstudio.com/search?term=pinvoke&target=VS&category=All%20categories&vsVersion=&sortBy=Relevance



P/Invoke를 사용하기 위해서는 먼저 System.Runtime.InteropServices 네임스페이스에 있는 DllImport를 사용하여 어떤 DLL(여기서는 kernel32.dll)에서 함수를 가져올지를 지정하고, 해당 함수의 원형을 C# 메서드 원형으로 정의해 준다. 이때 C# 메서드는 static extern 으로 지정한다.

```
using System;
using System.Runtime.InteropServices;
namespace Sample{
	class Program	{
		static void Main(string[] args)		{
			Beep(100, 1000);
			Beep(200, 1000);
			Beep(300, 1000);
			Console.ReadKey();
		}

		[DllImport("Kernel32.dll")]
		public static extern bool Beep(uint frequency, uint duration);
	}
}
```

예제)

C#에서 Win32 API를 호출하는 방법과 비슷하게, 자신이 만든 (혹은 3rd Party가 만든) C/C++ Native DLL 안의 함수를 호출할 수 있다. 이를 위해 먼저 (1) C/C++ 코드에서 해당 함수를 export 해야 하는데, 아래 예제에서 extern C 블럭이 해당 함수를 외부에서 사용할 수 있도록 export 하는 코드이다. 이러한 C/C++ 코드를 컴파일하여 DLL 로 만들어 두고, (2) 다음 C# 프로젝트에서 이 C/C++ DLL을 복사한 후, DllImport 를 정의하여 사용하면 된다. (주: 통상 C/C++ DLL은 C# 실행 파일과 같은 폴더에 두어야 한다.)

아래 예제는 ConvertFahrenheitToCelsius() 라는 C++ 함수를 C#에서 호출하여 사용하는 샘플이다.

```
#include "stdafx.h"

extern "C" {
    __declspec(dllexport) float WINAPI ConvertFahrenheitToCelsius(float f);
}

float WINAPI ConvertFahrenheitToCelsius(float f)
{
    float c = (f - 32) * 5.0 / 9.0;
    return c;
}

// C# 코드
using System;
using System.Runtime.InteropServices;
namespace UsePInvoke
{
    class Program    {
        static void Main(string[] args)          {
            float f = 82.50F;
            float c = ConvertFahrenheitToCelsius(f);

            Console.WriteLine("{0} C", c);
        }
        [DllImport("MyLib.dll")]
        public static extern float ConvertFahrenheitToCelsius(float f);
    }
}
```

예제)

Graphics.FromHwnd와 Graphics.FromHdc를 사용해 개체 얻기

```
public static Graphics FromHwnd( IntPtr hwnd);        // 창 핸들
public static Graphics FromHdc( IntPtr hdc);          // DC에 대한 핸들
```

```
using System;
using System.Drawing;
using System.Windows.Forms;
class GDIExam7 : Form
{
    Button btn1 = null;
    Button btn2 = null;
    public GDIExam7()
    {
        this.Text = "Graphics 개체 얻기7";
        btn1 = new Button();
        btn1.Text = "Graphics.FromHwnd 이용";
        btn1.SetBounds(10, 10, 200, 100);
        btn1.Click += new EventHandler(btn_Click);

        btn2 = new Button();
        btn2.Text = "Graphics.FromHdc 이용";
        btn2.SetBounds(10, 130, 200, 100);
        btn2.Click += new EventHandler(btn_Click);

        this.Controls.Add(btn1);
        this.Controls.Add(btn2);
    }

    static void Main(string[] args)
    {
        Application.Run(new GDIExam7());
    }

    [System.Runtime.InteropServices.DllImportAttribute("gdi32.dll")]
    private static extern bool Ellipse(
            IntPtr hdc,     // handle to DC
            int nLeftRect,  // x-coord of upper-left corner of rectangle
```

```
            int nTopRect,     // y-coord of upper-left corner of rectangle
            int nRightRect, // x-coord of lower-right corner of rectangle
            int nBottomRect // y-coord of lower-right corner of rectangle
        );

    public void btn_Click(object sender, EventArgs e)
    {
        if ((Button)sender == btn1)
        {
            IntPtr hwnd = new IntPtr();    // 핸들을 저장하는 레퍼런스 선언 및 생성
            hwnd = this.Handle;             //자신의 핸들 얻기

            Graphics grfx = Graphics.FromHwnd(hwnd);    //핸들을 이용하여 객체 얻기
            grfx.FillRectangle(Brushes.Blue, this.ClientRectangle);
            grfx.Dispose();
}
        else
        {
            //[C#]버튼에 Blue 색상으로 타원 출력
            Graphics g = this.btn2.CreateGraphics();
            g.DrawEllipse(Pens.Blue, 10, 10, 100, 70);

            //[Win32API]버튼에 Blue 색상으로 타원 출력
            IntPtr hdc = new IntPtr();
            hdc = g.GetHdc();
            Ellipse(hdc, 100, 10, 50, 50);

            g.ReleaseHdc(hdc);      //핸들 제거
            g.Dispose();
        }
    }
}
```

C++ <-> C# 간 타입 캐스팅

C++ 과 C# 타입 간 동일한 것이 없을 경우

Win32	비관리C 데이터타입	C#
HANDLE	int	int / IntPtr
BYTE	unsigned char	byte
SHORT	unsigned short	ushort
INT	int	int
UINT	unsigned int	uint / int
LONG	long	Int
BOOL	long	int
DWOD	unsigned long	uint
ULONG	unsigned long	uint
CHAR	char	char
LPSTR	char*	string / StringBuilder 리턴시 IntPtr
LPCSTR	const char*	string
LPWSTR	wchar_t*	string / StringBuilder 리턴시 IntPtr
LPCWSTR	const wchar_t*	string
FLOAT	float	float
DOUBLE	double	double