2장 : 아스키코드 vs 유니코드

Section 1. Windows에서의 유니코드

문자셋(Character Sets)의 종류와 특성

가장 대표적인 문자셋인 아스키 코드와 유니코드의 비교.

아스키 코드

* 미국에서 정의한 표준
* 알파벳 + 몇 개의 확장 문자를 포함한 문자셋.
* 1바이트로 한 문자를 표현 가능
* 떄문에 영어가 아닌 문자는 표현 불가능.

유니 코드

* 세계 표준
* 균일하게 모두 2바이트 사용
* 65536개의 문자 표현 가능.

문자셋이란 : 약속된 문자의 표현 방법. -> 종류에 따라 크게 세가지로 분류.

1. SBCS

* Single Byte Character Set.
* 문자 표현에 1바이트씩만을 할당. (아스키 코드)

1. MBCS

* Multi Byte Character Set.
* 문자 표현에 다양한 바이트 할당 : SBCS로 표현 가능한 문자는 1바이트, 그 외 문자는 2바이트 할당.

1. WBCS

* Wide Byte Character Set.
* 문자 표현에 일괄적으로 2바이트 할당. (유니 코드)

뭔가 MBCS로 문자를 표현하는 것이 합리적으로 보인다. 그러나, 프로그래머 입장에서 MBCS방식은 각 문자마다 할당된 크기가 다르므로 문제가 발생할 수 있다.

: 물론 프로그래밍으로 극복할 수 있지만, 그만큼 귀찮고 주의를 써주어야 한다.

* 이러한 문제점의 해결책으로 WBCS 방식을 사용할 수 있다.

WBCS 기반의 프로그래밍.

1. Char를 대신하는 wchar\_t

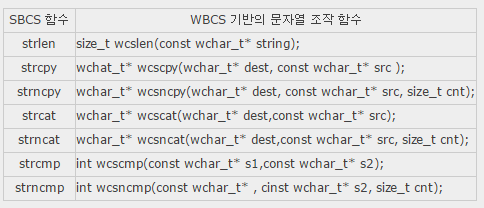
* char형은 메모리 1바이트만을 사용하여 공간을 할당하지만, wchar\_t는 2바이트 메모리 공간을 할당해준다.

1. “ABC”를 대신하는 L”ABC”

* “ABC”의 자료형은 const char이기 때문에 wchar\_t에 대입할 수 없다. 앞에 L을 붙여줌으로서 wchar\_t형으로 바꾸어서 대입해줄 수 있다.
* 유니코드에서는 널 문자까지도 2바이트로 처리가 된다. (즉, L”ABC”는 8바이트를 차지한다.)

1. Strlen을 대신하는 wcslen

* 앞과 마찬가지 이유로 유니코드를 지원하는 wcslen함수를 사용해야 한다.



* 기존에 사용하던 함수들은 싱글바이트 기준 함수들이므로, 유니코드 기반의 프로그래밍을 하기 위해서는 유니코드를 지원하는 함수로 바꿔줄 필요성이 있다. (함수 이름과 인자가 wchar\_t형으로 바뀌었을 뿐이다.)
* 다만 sizeof는 함수가 아니라 연산자이기 때문에 어디에 쓰더라도 올바르게 동작한다.

완전한 유니코드 기반으로

1. 윈도우 2000이상의 OS는 기본적으로 유니코드를 지원한다.

* 내가 SBCS 관련 함수를 호출하더라도 OS가 전달되는 문자열을 내부적으로 2바이트 유니코드 형식으로 변환하여 처리.

1. Main 함수를 대체할 wmain함수.

* 프로그램의 시작은 main함수부터 시작을 하는데, main함수의 인자는 char형이 포함 되어 있다.
* 그러므로 인자에 들어오는 char\*형을 wchar\_t\* 형으로 바꾼 wmain이라는 것이 존재한다.함수의 이름을 wmain으로 사용할 경우 main과 같이 프로그램의 시작점으로서 인식해준다.

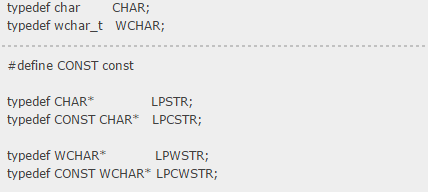
Section 2. MBCS와 WBCS의 동시 지원

그러면 모든 프로그램이 WBCS만을 사용하면 좋을텐데…

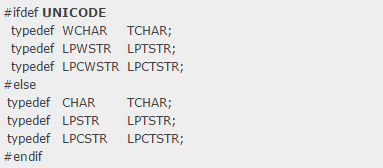
* 현존하는 시스템 모두가 유니코드 기반을 사용하는 것은 아님.
* 따라서 한 번의 프로그래밍에 WBCS와 MBCS방식 모두를 지원하는 방법이 필요했다.

자료형 동시 지원.

#include <Windows.h>에서 이러한 동시 지원을 위해 자료형을 정의해 놓았다.

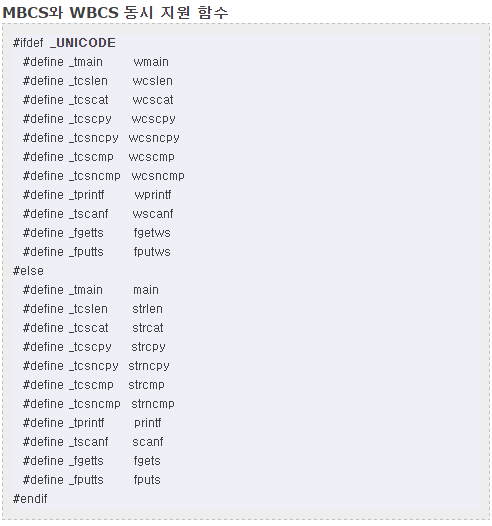


char형과 wchar\_t형을 각각 CHAR, WCHAR형으로 랩핑해두었다. 또한 이는 const 여부에 따라 LPSTR/LPWSTR, LPCSTR/LPCWSTR로 각각 랩핑이 되어있다. 그렇다면 이것이 무슨 소용인가?

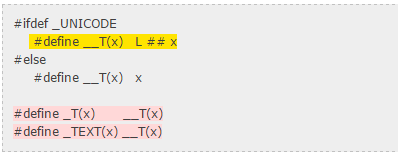


마찬가지로 <Windows.h>에 정의되어 있는 코드를 보면, UNICODE라는 매크로가 정의되어 있을 경우 TCHAR를 WCHAR 형태로, 아닐경우 TCHAR를 CHAR형태로 정의해 놓고 있다.

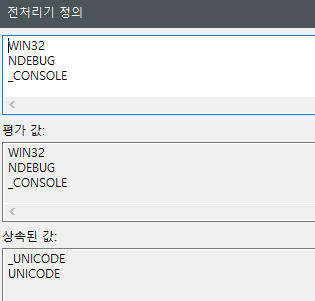
즉, TCHAR를 사용할 경우, 유니코드 기반의 프로그램이라면 wchar\_t형으로 변환이 되는 것이고 아닌경우 char형으로 변환이 되도록 자료형을 동시 지원해주고 있는 것이다. 이는 위에서 배운 함수 또한 마찬가지이다.



함수 또한 \_t를 이용한 함수를 정의해 두어서 유니코드 여부에 따라서 어떤 함수가 사용될지를 동시 지원해주고 있다.



문자열 표현 방식 또한 \_T(“ABC”) 혹은 TEXT(“ABC”)를 쓸 경우 동시 지원이 가능하다.



설정에서 전처리기쪽을 보면 어떤 값이 미리 정의 되어있는지 알 수 있다.