**조합형/완성형/유니코드의 모든 것**

|  |
| --- |
| 제목: 한글 및 한국어 정보 처리 코드(한국어 정보처리의 과제) 작성: 전상훈 (한국어정보처리연구소)날짜: 99.01.01.email: konan2@chollian.net klipl@klipl.com  |
|  |
| 배포자: 위형복■ 배포 목적: 한글의 조합형/완성형 등 한글 코드에 대한 지식과유니코드에 대한 전반적인 지식을 보급하고자 함.■ 배포자 연락처: dtpdosa@netian.com  |
|  |

이 글은 1998년 10월에 작성하여 1998년 11월 마이크로소프트웨어에 게재된 글이다. 마이크로소프트웨어 잡지에는 <표>와 <그림>이 많이 있지만 원래의 글에는 편집상 빠져 있다. 마이크로소프트웨어 잡지에는 따로 편집하여 <표>와 <그림>이 추가된 것이며, 이 글에서도 빠져 있다. 또한 마이크로소프트웨어 잡지에서 편집하는 과정에서 원래의 글과는 약간의 차이가 발생하였는데, 이 글도 마이크로소프트웨어 잡지에 게재된 내용과는 약간의 차이가 있음을 밝힌다.

(월간 마이크로소프트 1998년 11월호 특집 참조 요망)

▶ 차 례 ◀

[■ 들어가는 말 - 뾼차를 타고 온 최시맨과 뉵다리 馱방각하의 만남](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 들어가는 말 - 뾼차를)

[■](file:///F%3A%5CSinDisk%5CTxtnote%5C%EA%B8%B0%ED%83%80%EA%B8%80%5CUnicode%5C%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#■ 한글 코드의 정의) [한글 코드의 정의](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 한글 코드의 정의)
....... [● 한글 코드의 종류와 원리](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 한글 코드의 종류와 원리)
.............. [▶ N바이트 한글 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ N바이트 한글 코드)
.............. [▶ 3바이트 한글 코드와 자소형 한글 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ 3바이트 한글 코드와 자)
.............. [▶ KSC5601-1987완성형 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ KSC5601-1987완)
.............. [▶ 상용 조합형 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ 상용 조합형 코드)
.............. [▶ KSC5657-1991 확장 표준 코드(필자가 편의상 지정한 이름)](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ KSC5657-1991)
.............. [▶ KSC5601-1992 조합형 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ KSC5601-1992)

[■ 잘 못 끼어진 첫 단추는 이렇게 끼워졌다.](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 잘 못 끼어진 첫 단추는)
....... [● KSC5601-1987이 보급되는 과정](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● KSC5601-1987이)

[■ KS완성형 코드의 문제점](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ KS완성형 코드의 문제점)

[■ 확장 완성형 코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 확장 완성형 코드)
....... [● 확장 완성형 코드가 지닌 한계와 문제점](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 확장 완성형 코드가 지닌)
....... [● 확장 완성형으로 한글을 표현하는 데 문제가 있는가?](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 확장 완성형으로 한글을)
....... [● 사실상 표준으로 영향력을 미치게 될 확장완성형](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 사실상 표준으로 영향력을)

[■ 유니코드에서 한글 코드 배정](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 유니코드에서 한글 코드)
....... [● 유니코드 일반적인 구조와 한글 처리 원리](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 유니코드 일반적인 구조와)
....... [● 유니코드에서 한자는 어떻게 되었는가](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 유니코드에서 한자는 어떻)
....... [● 유니코드는 한글 및 한국어를 완벽하게 처리할 수 있는가](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 유니코드는 한글 및 한국)

[■ 한글 윈도우98과 문서처리기에서 한글 코드 및 한글 처리](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 한글 윈도우98과 문서처)

[■ 한글 표준 코드는 어떻게 처리되어야 하는가](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 한글 표준 코드는 어떻게)

[■ 코드 변환에 대하여](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 코드 변환에 대하여)
....... [● KS완성형 <-> 조합형(상용)](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● KS완성형 <-> 조합형)
....... [● KS완성형 <-> 유니코드](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● KS완성형 <-> 유니코)
....... [● 조합](file:///F%3A%5CSinDisk%5CTxtnote%5C%EA%B8%B0%ED%83%80%EA%B8%80%5CUnicode%5C%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#● 조합형 <-> 유니코드)[형](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#● 조합형 <-> 유니코드) [<-> 유니코드](file:///F%3A%5CSinDisk%5CTxtnote%5C%EA%B8%B0%ED%83%80%EA%B8%80%5CUnicode%5C%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#● 조합형 <-> 유니코드)

[■ 정보는 자원이자 곧 국가 경쟁력이다.](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#■ 정보는 자원이자 곧 국가)

▶ **박스기사**
[▶ [박스] ISO-2022 규격](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ [박스] ISO-2022)
[▶ [박스] Unicode와 ISO-10646의 관계](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ [박스] Unicode와)
[▶ [박스] KSC5700코드와 문제점 - 제 1차 한글 코드 표준화 위원회 회의록](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ [박스] KSC5700코)
[▶ [박스] 유니코드에서 현대 한글 완성형 11172자를 획득하는 과정](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ [박스] 유니코드에서 현)
[▶ [박스] 한글 코드별 효율성 비교 검토](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ [박스] 한글 코드별 효)
[▶ 참고 문헌](http://www.copyeditor.co.kr/reference/char-font/johabwansung.htm#▶ 참고 문헌)

■ 들어가는 말 - 뾼차를 타고 온 최시맨과 뉵다리 馱방각하의 만남

정보 검색이나 인터넷 검색이라는 말이 일반인에게 알려진 것은 최근 2 ~ 3년도 채 되지 않았다. 그럼에도 불구하고 이제는 어린 유치원생부터 중년의 주부나 노년의 할아버지까지 인터넷을 따라잡기에 여념이 없다. 아마도 인터넷이 지닌 무한한 잠재성과 정보화 사회로 가는 길목에서, 정보를 어디서 얼마나 어떻게 획득하느냐가 생존을 좌우하기 때문일 것이다. 이러한 와중에서 필자에게 문득 재미있는 생각이 떠올랐다. 한때 폭발적인 인기를 끌었던 펩시콜라 광고에 나오는 최시맨과 드라마로 유명했던 馱방각하가 팔씨름을 한다면 과연 누가 이길지 궁금했다. 그래서 먼저 이들이 있을 만한 인터넷 홈페이지를 찾아보았는데 뜻밖에도 찾을 수가 없었다. 펩시맨은 있어도 [최시맨]은 찾을 수가 없었고, [馱방각하]를 찾으면 대부분 찾지 못했으며 설령 찾았다 하더라도 소가 뒷걸음 치다가 쥐잡는 격으로 얼떨결에 찾는 것을 보고서 놀라지 않을 수 없었다. 검색 결과를 보면, 시스템 장애가 발생하였으므로 회사로 연락을 하라고 하던가, 다른 단어는 멀쩡한데 이런 단어는 바빠서 항상 처리할 수 없다든가, 아니면 이런 검색어는 문법 오류가 있다고 답하던가, [馱방각하]는 [饯방각하, 깎방각하]로 바꾸어서 돌려보내고, [최시콜라]는 [钰시콜라]로 바꾸어서 보내면서 찾을 수가 없다는 것이다. 아주 재미있는 결과이지만 그다지 유쾌하지는 않았다. 필자가 검색했던 인터넷 검색 엔진들은 한글 처리 부문에서 자타가 공인하는 최고 수준의 회사라는 점에서 어떻게 이런 일 있을 수 있는지 매우 궁금했다. 아마도 이러한 문제는 필자뿐만 아니라 독자에게도 궁금하게 다가올 듯한 문제이므로 그 이유를 더듬어 보기 위해서 글을 쓰게 되었다.

이러한 문제는 한글 코드와 밀접한 관련이 있기 때문에, 먼저 한글 코드가 무엇이며, 그러한 코드는 어떻게 변해왔는지 간단하게 살펴보고, 각각의 특징을 살펴보면서 윈도우95에 이은 윈도우98의 등장은 어떠한 관계가 있으며, 앞으로 발전 방향은 무엇인지 함께 생각해보자.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 한글 코드의 정의

컴퓨터는 내부적으로 정보를 처리하기 위해서 2진수 형태로 처리되기 때문에 인간이 사용하는 언어로는 사용할 수가 없다. 그래서 인간의 언어를 컴퓨터가 사용할 수 있도록 이진수의 대응 관계를 정의한 것을 문자 코드라고 한다. 따라서 한글 코드라는 것은 한글 및 한국어를 컴퓨터 내부에서 이진수로 처리하도록 정의해 놓은 문자 집합이다. 여기서 한글 및 한국어라고 하는 이유는 한글과 함께 사용하는 숫자, 영문자, 한자, 기타 등등의 문자를 포함하기 때문이다. (앞으로는 간단하게 한글 코드라고 부르겠다.)

초기의 컴퓨터는 영어권 문화에서 만들어진 기계였으므로 컴퓨터를 쉽게 다루기 위해서 영어로 된 명령을 사용하도록 제작되었다. 이러한 컴퓨터가 국내에 처음 들어왔을 무렵에는 소수의 전문 교육을 받은 사람들만이 컴퓨터를 다루었기 때문에 영어를 그대로 사용했다. 그러다가 컴퓨터의 보급이 점차 증가되기 시작하면서 한국어로 된 명령과 한국어를 처리할 수 있는 프로그램이 필요하게 되었다. 그런데 그 당시 컴퓨터로는 한국어를 처리하려면 여러 가지 어려운 문제가 있었는데, 가장 커다란 문제는 기본적으로 컴퓨터는 1바이트 체계로 되어 있었기 때문에 최대한 표현할 수 있는 문자는 256자였다는 점이다. 현대 한글은 11172자였으므로 256자까지 표현할 수밖에 없는 1바이트 체계로는 현대 한글을 전혀 표현할 수가 없었다. 그래서 능력있는 개인이나 기업들이 하나 둘씩 각각의 한글 처리 방법을 고안하게 되었다. 이때까지만 해도 컴퓨터에서 한글을 사용한다는 것 자체가 대단한 일이었으므로 처리 방법이나 구현 원리 같은 것은 중요한 문제가 아니었다.

그림. 한글 코드의 예

|  |  |  |
| --- | --- | --- |
| 문자 | 이진수 | 십육진수 |
| 가 | 1011000010100001 | B0A1 |
| 각 | 1011000010100010 | B0A2 |
|  |

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 한글 코드의 종류와 원리

현재 국내에서 사용하거나 사용되었던 한글 코드는 많이 있지만 현실적으로 지금 사용하거나 앞으로 사용하게 될 코드 혹은 이미 사용되었지만 중요한 의의를 지닌 것들을 중심으로 설명하는데 한글 코드를 분류하는 기준과 각각의 한글 코드에 대하여 알아보자.

한글 코드를 분류하는 기준은 여러 가지가 있지만 가장 중요한 기준은 한글 한 음절을 표현하는데 소요되는 바이트 수와 8번째 비트를 사용 여부이며 지금까지 제정되거나 사용되었던 것은 여러 가지가 있었지만 현실적으로 사용하거나 중요한 역할을 하는 것만 다루겠다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ N바이트 한글 코드**

이것은 한글을 풀어 쓴 것과 같이 각각 자음과 모음을 한 바이트씩 처리하는 것이다. 따라서 한글 한 음절을 표현하기 위해서는 길이가 2바이트부터 5바이트까지 사용한다. 그래서 보통은 멀티바이트(multi-byte)코드라고도 한다. 이것은 80년대 초반에 널리 사용되었으며 7비트 아스키 환경에서도 사용할 수 있다는 장점이 있지만 한글 처리에서는 적합하지 않아서 곧바로 사라지게 된다. 무엇보다도 가장 문제가 되었던 것은 문자를 비교하거나 순서대로 정렬(sort)할 경우에 한글 순서와 맞지 않는다는 것이다. 이렇게 문제가 많아서 사라진 코드를 설명하는 데에는 다른 이유가 있다. 그것은 현재에 정보 검색, 맞춤법 검사, 자동 색인, 형태소 분석 등등에서는 N바이트 코드방식을 이용하고 있다는 점이다. 그 이유는 완성형이나 조합형 코드로는 언어 정보를 처리할 수 없기 때문인데, 현대어는 물론이고 옛한글은 더욱 불가능하다. 물론 유니코드는 현대어와 옛한글을 거의 모두 처리할 수 있도록 되어 있으므로 유니코드에서 다시 설명하겠다.

그림. N바이트 한 음절 구성 예

나 = ㄴ + ㅏ => 2바이트

닥 = ㄷ + ㅣ + ㄱ => 3바이트

삶 = ㅅ + ㅏ + ㄹ + ㅁ => 4바이트

觫 = ㄱ + ㅗ + ㅏ + ㄹ + ㅁ => 5바이트

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ 3바이트 한글 코드와 자소형 한글 코드**

3바이트 한글 코드는 한글 한 음절을 초성, 중성, 종성으로 나누고 각각 한 바이트씩 배정하여 처리하는 것으로 한글 한 음절을 처리하기 위해서 3바이트를 사용하여 붙여진 이름이다. 이것도 일종의 조합형 코드이지만 한글 1음절을 표현할 때 2바이트 조합형 코드보다 1바이트를 더 많이 사용하기 때문에 비효율적이어서 자취를 감추게 된다.

자소형 코드는 특별히 제정되었거나 널리 사용된 코드는 아니지만 필자가 편의상 설명하는 코드이다. 이 코드는 한글을 표현하기 위해서 자소 단위로 처리하는 것으로 N바이트 한글 코드와 거의 똑같으며 조합형 코드로는 처리할 수 없는 언어 정보를 처리하기 위해서 필요에 따라 만들어 사용하는 코드이다. 이 코드를 설명하는 이유는 현재 인터넷 정보검색, 맞춤법 검사기, 자동 색인 등에서 이 방식을 거의 사용하고 있기 때문이다. 이 코드는 사용하는 원칙이나 방법이 공식적으로 제정된 적은 없고 필요한 분야에서 각자 만들어 사용하고 있는 실정이며, 유니코드가 제정되면서 현실화되기도 한다.[그림. N바이트 코드 참조]

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ KSC5601-1987완성형 코드**

KSC5601에서 채택한 방식을 한글을 음절 단위로 순서대로 배치하여 각각의 코드값을 부여하는 방식이다. 구현 방식을 보면 첫 번째 바이트와 두 번째 바이트 모두 164 ~ 256까지의 영역만을 사용한다. 그래서 실제로 사용 가능한 문자는 94 \* 94(8836)자이므로 현대 한글 11172자를 모두 표현할 수가 없는 문제점이 발생한다. 이와 같은 문제를 피하기 위해서 많이 사용하는 한글 음절 2350자, 한자 4888자, 특수문자 1128자, 나머지 470자를 배정한다. [그림. KSC5601코드의 글자 배치]

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ 상용 조합형 코드**

한글 한 음절을 초성, 중성, 종성으로 나누어 각각 5비트씩 배정하고 7비트 아스키코드에서 사용하지 않는 최상위 비트(MSB)에 '1'로 배정하여 2바이트로 처리한다. 첫 번째 바이트 최상위 비트가 '1'로 되어 있으면 한글 한 음절로 해석하고, '0'으로 되어 있으면 영문자로 구분한다. 그런데 조합형 코드는 회사별로 서로 다르게 제정하였으나 KS완성형 코드가 제정되면서부터 대부분의 조합형 코드는 자취를 감추게 되었고, 상용 조합형(삼보 조합형 혹은 KSSM 조합형)만이 표준 코드 제정 이후에도 계속해서 사용되었으므로 대개 조합형 코드라는 것은 이것을 가리킨다.[그림. 조합형 코드]

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ KSC5657-1991 확장 표준 코드(필자가 편의상 지정한 이름)**

KSC5601-1987완성형 코드의 제정과 함께 불거진 코드에 대한 문제점을 보완하기 위해서 1991년에 만들어진 것으로, 한글 1930자, 한자 2856자, 옛한글 1677자 등이 추가되었다. 그런데 앞서 표준으로 제정되었던 KSC5601 완성형 코드를 부분적으로 보완하였기 때문에 현대 한글 6892자는 빠져버린다. 하지만 확장 표준 코드에서 한자가 7744자로 늘어나고 옛한글이 표준 코드에 반영되었다는 점이 중요하다. 실제로 이 코드는 전혀 사용하지 않았기 때문에 확장 코드를 기억하는 사람이 없지만 나중에 한자와 옛한글을 표준 코드에 다루는 데 중요한 단서가 된다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ KSC5601-1992 조합형 코드**

KSC5601-1987완성형 코드에서는 모든 현대 한글을 표현할 수 없다는 점에서 민족 문화의 계승 발전과 산업 및 기술적 측면에서 많은 문제점을 노출시켰다. 그래서 정부는 현대 한글을 완벽하게 표현할 수 있는 조합형의 필요성을 인식하여 1992년에 이르러 또 하나의 표준 코드(KSC5601-1992)를 제정하게 되는데, 이로 인하여 표준 한글 코드가 두 개가 되어 버린다. 또한 이미 KS완성형 코드로 작성된 문서와 데이터와는 호환성이 없어서 현실적으로는 거의 사용하지 않게 된다. 이 때 제정된 조합형은 상용 조합형 코드와 구성이 거의 비슷하지만 몇 글자의 코드는 다르게 배정되어 있으며, 현재의 몇몇 응용 프로그램에서는 상용 조합형 대신에 표준 조합형 코드를 지원하고 있는 실정이다.(1987년을 기점으로 주로 사용하는 조합형을 상용 조합형 코드라고 하며, 1992년에 제정된 조합형 코드를 표준 조합형 코드라고 한다.)

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 잘 못 끼어진 첫 단추는 이렇게 끼워졌다.

컴퓨터에서 한글 코드 문제를 이해하기 위해서는 한글 코드에 관한 역사적인 흐름을 이해해야 하는데, 한글 코드가 실질적인 표준으로 제정되는 1987년 전후에 상황을 살펴보고, 이어서 한글 코드의 정의와 이미 사용하거나 사용했던 한글 코드의 특징을 차례대로 알아보겠다.

1974년 처음으로 한국 공업 규격 KSC 5601-1974가 제정되었으며, 뒤이어 1977년에 KSC 5714-1977이 제정되었다. 이 때는 한글 자모 51자와 한자 7200자를 정해놓았으며 특히 자모에 의한 처리는 한계가 많았기 때문에 조합형과 완성형 코드가 등장하게 된다. 완성형 코드로 된 KSC 5619-1982와 조합형으로 된 KSC 5601-1982가 생겨났다. 여기서 중요한 점은 각각의 코드가 국가 표준으로 제정이 되었다는 뜻이지 실제로 모두가 이렇게 사용했다는 뜻은 아니다. 당시의 컴퓨터 제조 업체는 각각의 한글 코드 체계를 사용했으며 대부분의 회사들은 조합형의 장점 때문에 조합형을 사용하고 있었다. 그러한 가운데 개인용 컴퓨터가 대중화되면서 한글 코드 문제도 본격적으로 불거지기 시작했다. 예를 들어, A라는 회사에서 사용하던 한글 문서(코드)는 B라는 회사 컴퓨터에서는 사용할 수 없고, 역으로 B에서 사용하던 한글 문서(코드)는 A라는 회사에서 사용할 수 없게 되어 버린 것이다. 이와 같은 한글 코드간의 호환성이라는 문제가 대두되면서 나름대로 노력을 시도하였지만 별다른 효과가 없었다. 이 때까지만 해도 소프트웨어는 하드웨어를 팔기 위한 하나의 수단이었으므로 일단 컴퓨터를 많이 팔아서 세력을 얻는 방법이 효과적이라고 여겼기 때문에 해결이라는 것은 애당초 생각조차 하기 힘들었던 시기였다. 시간이 지날수록 불편함이 가중되었고 그만큼 사용자들의 표준화 요구도 더욱 절실해졌다. 이러한 문제를 해결하기 위해서 업체 중심으로 한글 코드 표준화를 시도하게 되었다. 처음에는 대부분 조합형 코드를 사용하고 있었으므로 완성형보다는 조합형을 지지하게 되었다. 그러나 각각 자신의 코드를 중심으로 표준화를 시도하다 보니 아무런 결말도 낼 수 없었는데, 자신의 코드를 포기할 경우 이전에 판매했던 제품과의 호환성이 문제가 될 뿐만 아니라 판매에도 중요한 영향을 미치기 때문에 사실상 다른 회사의 코드를 지지할 수가 없었다. 당시 이십여 개의 회사가 참여했으며 시간이 지날수록 조합형 코드를 포기하는 쪽으로 결론이 나면서 최종적으로 완성형을 채택하게 되었다.(앞으로는 KSC5601-1987내지는 KS완성형 코드라고 부르겠다. 이전의 완성형 코드와는 차이가 있으며 현재의 표준 코드는 조합형도 채택되어 있기 때문에 혼동할 우려가 있으므로 주의하기 바란다.) 물론 이 때 완성형을 채택하게 되는 아주 중요한 이유가 있다. 그것은 국제 표준화 기구(ISO)에서 정한 국제 규격과 호환성 때문이었다.(ISO-2022참조)

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● KSC5601-1987이 보급되는 과정

우여곡절 끝에 1987년에 완성형을 표준으로 하는 KSC5601-1987(혹은 KS완성형 코드)이 제정되었어도 상당수의 컴퓨터 및 소프트웨어 제조 회사들은 계속해서 조합형을 사용하다가 국가에서 주도한 행정 전산망 사업에서 KS완성형을 기본 코드로 채택하면서부터 점차 보급되기 시작한다. 게다가 뒤이은 교육용 컴퓨터 보급 사업에서는 행정 전산망과 호환되는 코드를 사용한다는 원칙을 제시되면서부터 본격적으로 KS완성형 코드가 보급되었다. 더욱이 개인용 컴퓨터의 운영 프로그램인 DOS를 제작하였던 마이크로소프트사가 1989년에 이르러서는 조합형을 포기하고 KS완성형을 지원하였으며, 중대형 컴퓨터의 운영 프로그램인 UNIX 진영에서도 KS완성형을 공식적으로 지원하면서 사실상 표준으로 자리잡아 가게 된다.

여기서 짚고 넘어가 할 중요한 부분이 있다. 현재 한국은 물론 전 세계적으로 개인용 컴퓨터 운영 프로그램 판매에서 막대한 시장 점유율을 자랑하는 마이크로소프트사는 그 당시에 조합형을 사용하고 있었으며, 국가적 표준 코드(완성형) 보급 정책과 맞물려 조합형을 포기하였다는 점이다. 그러다가 1995년 윈도우95를 발매하면서 확장(통합) 완성형 코드를 제정하여 발표하는 과정에서 여론의 비판적인 화살을 맞으면서 완성형 코드가 마치 마이크로소프트사의 산물인 양 비추어졌던 점을 상기하면서 주목하기 바란다.

또 한가지 덧붙여 주목할 점은 그 당시 한글 코드 표준화에 참여하고 오늘날까지도 컴퓨터를 판매하는 회사가 KS완성형 코드를 채택하였다는 것은 만들어냈다는 뜻과는 다르다는 것이다. 말뜻 그대로 통일된 표준 코드로써 채택된 완성형 코드를 수용하는 동시에 보급하는 역할을 하게 되었던 것이고, KS완성형 코드를 제정하는 순간에는 전산 전문가와 통신 전문가들만 참여했으며, 국어학자나 출판 문화 사업에 관계된 종사자는 없었다는 점이다.

KS완성형이 표준 코드로 보급되면서 서서히 논쟁이 불붙기 시작한다. 1989년에 이르러서는 컴퓨터 전문 잡지를 비롯하여 일간지, 컴퓨터 통신 등을 통해서 KS완성형 코드의 문제점이 제기된다. 이 때, 제기된 문제점은 여러 가지가 있지만 무엇보다도 글자를 모두 표현할 수 없다는 점이 가장 큰 문제였다. 이렇게 문제가 불거져 나오면서 KS완성형 코드를 보완하는 개정작업에 들어갔으며 그 결과 1991년 12월에 KSC5657을 제정한다. 그러나 이것마저도 충분히 검토하지 않고 제정됨으로써 사실상 보급되지 않게 된다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ KS완성형 코드의 문제점

KS완성형의 문제점을 설명하기 전에 먼저 한글 코드가 지녀야 하는 기본적인 원칙이 있다.

무엇보다도 한글 특성을 반영해야 한다는 점이다. 표현 가능한 문자는 모두 처리할 수 있어야 하며, 옛한글도 현대어처럼 다룰 수 있어야 하고, 비한글 문자(한자, 영문자, 일문자, 기타 문자)도 사용하는데 문제가 없어야 한다. 그 다음으로는 기본적으로 7비트 아스키코드로 동작하는 영문 자료와 제어 문자들과는 충돌하지 말아야 하고, 보다 중요한 정보 처리를 위해서는 음절을 넘어서 자소 단위로 처리할 수 있어야 한다. 끝으로 국제적인 표준 규격을 따라야 하며, 산업 현장에서 현실적으로 수용할 수 있어야 하며, 정부의 정책적인 지원이 있어야 한다.

이러한 기본적인 원칙을 중심으로 KS완성형을 살펴 보면 다음과 같은 문제점이 발생한다.

가장 중요한 것은 표현 가능한 현대 한글을 모두 표현할 수 없었으므로 한글 특성을 살리지 못 한다는 점이다. KS완성형 코드는 현대 한글 11172자 중에서 2350자만을 처리할 수 있기 때문에 문제가 발생한다. 간단히 예를 들자면, "최시콜라, 최시맨, 馱방각하, 뾼차, 뉵다리"등을 표현할 수 없다. 또한 '쓩'이라는 자는 KS완성형 코드에는 있지만 '덁'라는 자가 없어서 입력할 수 없는 경우가 6자나 발견되었다. 이것은 KS완성형 코드를 표준으로 제정하는 과정에서 실제로 사용할 수 있는가에 대한 타당성을 검토하지 않은 결과였으므로 당시에는 대단한 비웃음거리가 되었다. 이것말고도 자소 정보를 추출할 수 없었으므로 한국어에 대힌 언어적인 정보를 처리하기 위해서는 조합형을 거치게 하는 결과를 낳았다. (조합형 코드를 거쳐야만 자소 단위의 정보를 처리하기 쉽기 때문이다.)

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 확장 완성형 코드

이것은 마이크로소프트사가 독자적으로 제정하여 사용하는 것으로 윈도우95와 함께 널리 사용되게 된다. 기본적으로는 KS완성형 코드을 그대로 사용하고 KS완성형 코드에서 표현할 수 없었던 한글 8822자를 KS완성형 코드에서 사용하지 않는 영역에 추가로 배정한 방식이다. 이 방식을 사용함으로써 조합형과의 논쟁에서 골칫거리였던 나머지 한글을 모두 표현하게 되었다는 점이 매우 중요하다. 그럼 마이크로소프트사의 당시에 발표한 내용을 보면 다음과 같다.[그림. 확장 완성형 코드 영역]

"마이크로소프트사는 1992년 조합형 발표 이후 모든 한글을 포함하는 조합형(KSC5601-1992)을 지원하기 위해서 내부적으로 많은 노력을 해왔으나, 기존의 대부분 완성형만을 지원하는 프로그램과 자료 데이터와의 심각한 호환성 문제가 항상 걸림돌로 되어 왔다. 이에 마이크로소프트사는 기존의 프로그램 및 데이터와의 호환성을 유지하면서 모든 한글을 지원하는 데 가장 바람직한 방법은 완성형(KSC5601-1987) 코드를 그대로 두고, 나머지 한글을 남아 있는 DBCS(Double Byte Character Set) 영역으로 확장하는 방법으로 확장 완성형 코드를 개발하였다."

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 확장 완성형 코드가 지닌 한계와 문제점

확장 완성형 코드가 지닌 한계를 알기 전에 이것을 제정하게 되는 이유와 원칙을 알아 보아야 한다. 그래야만 한계와 문제점을 쉽게 이해할 수 있기 때문이다. 확장 완성형을 제정하게 되는 가장 중요한 이유는 KS완성형 코드에서 처리할 수 없었던 현대어 8822자를 처리하기 위한 것이고, 제정 원칙은 기존의 완성형만을 지원하는 프로그램 및 데이터와의 호환성을 유지한다는 것이다. 여기서 확장 완성형이 비판을 받게 되는 이유는 호환성을 위해서 한글 순서(사전 순서)를 무시한 코드 배정에 있었다. [그림. 확장 완성형 코드 영역 참조]

즉, 기존의 KS완성형 2350자는 '가나다'순으로 배열되어 있으나, 나머지 글자들이 그 위(앞)에 배치됨으로써 전체 코드 내에서 한글의 순서가 뒤죽박죽 엉키게 되며, 정렬(sorting)이나 탐색(search)을 할 때 심각한 영향을 미치기 때문이다. 물론 이러한 문제는 개발자가 조합형으로 변환하여 사용하면 간단하게 해결할 수도 있다.

그런데 더욱 논란이 되었던 것은 확장 완성형 코드 자체보다는 확장 완성형 코드를 발표하는 과정과 그것이 미칠 영향력 때문이었다. 확장 완성형 코드는 어느 한 기업의 제품일 수도 있지만 국내 산업 현실을 고려해보면 사실상 표준 코드로써 막대한 영향력을 미칠 수도 있는 상황이었으므로 예상 밖의 커다란 논란이 되었다. (여기서는 당시의 논쟁을 언급하지 않겠다. 독자들 중에서는 논쟁에 대하여 궁금하게 생각하는 독자도 있겠지만 이 글의 목적은 논쟁을 재현시키는 것이 아니라 지나온 과정을 살펴보고 발전적인 방향을 검토하는 것이므로 관심있는 독자들은 참고 자료를 보기 바란다.)

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 확장 완성형으로 한글을 표현하는 데 문제가 있는가?

먼저 결론부터 말하자면 확장 완성형 코드로 한글을 어느 정도는 처리할 수 있다. 좀더 구체적으로 말해서, 미완성 음절 문자 1147자(예: 葶, 蓶, 끁)를 처리할 수 없다는 점을 빼고는 조합형만큼 처리를 잘 할 수 있다. 그러나 제대로 처리하기 위해서는 경우에 따라서 조합형으로 변환해야 하기 때문에 그에 따른 컴퓨터의 연산 속도에 영향을 미치며 이것이 전체적으로는 비효율적이라고 비판하는 사람이 적지 않다. 그런데 필자의 개인적인 생각으로는 이 정도의 연산 속도가 문제가 될 정도라면 컴퓨터는 아예 사용하기도 힘들 정도이다. 오히려 전체적인 작업에 비하면 조합형 변환에 따른 연산 속도는 거의 영향을 미치지 않는다고 보아야 할 듯하다. 전체적인 작업 시간과 비교하였을 때 문서처리기, 정보 검색, 맞춤법 검사, 으뜸꼴 인식, 기타 다른 작업에서 조합형으로 변환하는 시간은 거의 영향을 미치지 않는다고 말할 수 있을 정도이며, 실제로 조합형 코드도 한국어 정보 처리를 위해서는 자소형 코드로 변환을 거치기 때문에 똑같이 연산 속도에 영향을 미친다는 것을 생각하면 확장 완성형 코드를 가지고 정렬과 탐색에서 조합형 코드로 변환하는 시간을 문제 삼기에는 다소 이해하기 어려운 부분이다. 그러나 다른 한편으로는 정보 검색, 자동 색인, 으뜸꼴 인식, 맞춤법 검사기에서는 조합형 코드를 그대로 사용할 수가 없어서 자소형으로 변환하여 사용하고 있다는 점을 생각해보면 확장 완성형은 두 번의 변환 과정을 거쳐야 하기 때문에 그만큼 비효율적인 것은 사실이다. 어쨌든 확장 완성형으로 현대어를 입력하고 출력하는 데는 문제가 없으며, 사전식 순서를 원할 경우에는 조합형으로 변환시켜 사용하면 별다른 문제가 되지 않는다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 사실상 표준으로 영향력을 미치게 될 확장완성형

필자의 개인적인 생각으로는 확장 완성형이 원래 의도하지 않았던 문제가 발생하고 있다. 이 글을 시작할 때 소개하였던 "뾼차를 타고 온 최시맨과 뉵다리 馱방각하"를 인터넷에서 찾으려면 심각한 문제가 발생한다. 앞서 지적했듯이 거의 모든 정보 검색 시스템이나 인터넷 관련 프로그램은 이것을 제대로 처리하지 못하고 있다. 심지어는 정보 검색 시스템에서 장애가 발생하였다거나 아예 글자를 엉뚱하게 바꾸어서 처리하는 경우도 있으며, 인터넷 관련 프로그램에서는 이상한 박스 문자로 나오거나 비어 있는 경우가 발생한다. 이것은 대부분의 프로그램이 기본 완성형(KS완성형) 코드만을 고려하여 개발되었기 때문에 기본 완성형 코드에 없는 확장 완성형 한글 문자를 처리하면서 문제가 발생하는 것으로 해결 방법은 크게 두 가지이다. 하나는 기본 완성형 한글 문자만 처리하고 확장 완성형 한글은 막아 버리는 것이고, 또 다른 하나는 확장 완성형 한글 문자를 추가로 처리하여 문제를 해결하는 것이다. 첫 번째의 경우는 코드 변환 부분에서 막는 것으로 해결하기 때문에 간단하며, 두 번째의 경우도 확장 완성형 한글 문자 영역에서 한글 문자를 추가로 다루면 별다른 문제 없이 해결할 수 있다. 바로 이 점이 우리가 주목해야 하는 부분이다. 별다른 어려움 없이 확장 완성형 한글 문자를 처리할 수 있으므로 새로운 표준 코드가 정착되기 전까지 대부분의 정보 검색 시스템에서는 확장 완성형 코드를 지원할 가능성이 있다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 유니코드에서 한글 코드 배정

유니코드는 매우 중요한 의미를 지니고 있기 때문에 유니코드의 출현 배경을 알아 보고, 유니코드에서의 한글 코드 변천과 유니코드에서 한글 코드의 특징을 살펴 보자.

유니코드는 국제 표준화 기구는 아니며 컴퓨터 관련 업계를 중심으로, 다국어 언어를 효율적으로 처리에 필요한 코드 체계를 제정하기 위해서 1989년에 콘소시엄 형태로 설립된 회사이다.

1991년 말에 유니코드 1.0을 발표하였는데, 여기서 한글 코드는 KSC5601-1987완성형 코드만을 포함시켰기 때문에 국내에서는 환영을 받지 못했다. 그래서 뒤이은 유니코드1.1. 개정에서는 조합형 코드를 수용해줄 것을 강력하게 요청하였지만 부분적으로만 수용하는 바람에 실제로는 전혀 사용할 수조차 없는 얼치기 한글 코드가 배정되었다. 당시에 한글 코드를 조합형으로 배정받지 못 했던 중요한 이유는 한글 조합형 코드가 다른 나라에 비하여 너무 많은 영역을 요구하였기에 심하게 반발하였던 것이다. 한편으로는 조합형 코드를 조합 방식 그대로 적용하면 대략 3만 자가 넘기 때문에 수용하기 어려운 면이 있었으며, 조합 규칙을 간소화하여 적용하여도 최소한 11172자를 사용해야 했으므로 당시로는 국제 사회에서 받아들이기 어려웠다. 이 때까지만 해도 전세계적으로 유니코드는 현실적으로 사용하지 않았기 때문에 1995년에 이르러 유니코드2.0을 제정하게 된다. 유니코드2.0에서는 한글은 두 가지 영역으로 배정받았다. 첫 번째는 현대 한글 11172자를 완성형 방식의 코드순으로 배열한 것인데, 현대 한글 11172자는 완성형처럼 완성된 음절을 기준으로 배정하였기 때문에 완성형 방식 코드라고 부르며(이 글에서는 앞으로 혼동되지 않도록 유니완성형으로 사용) KSC5601완성형 코드와는 다르게 일정한 조합 규칙을 유지하고 있어서 사용하기 편리하다. 두 번째는 초성, 중성, 종성을 자소 단위로 배정한 조합형 방식 코드로(유니코드에서는 한글자모[Hangul Jamo]라고 하지만, 이글에서는 앞으로 혼동되지 않도록 유니조합형으로 사용) 2바이트 상용 조합형이나 KSC5601-1992 표준 조합형처럼 고정된 길이가 아니라 자소 개수만큼 사용하는 N바이트 형식으로 지정하였다. 현재에는 유니코드 2.1이 최종 버전으로 발표되었으며, 국내에서는 유니코드 2.0이 발표되자마자 KSC5700이라는 이름으로 유니코드 2.0을 국가 표준 코드로 채택하게 된다.(유니코드 2.0과 KSC5700은 사실상 같으므로 같은 의미로 사용한다.)

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 유니코드 일반적인 구조와 한글 처리 원리

유니코드에서 한글 처리 원리를 알기 전에 일반적인 원리와 알아볼 필요가 있다. 유니코드는 기존의 아스키 코드, KS완성형, 조합형(상용, 표준)형과는 근본적으로 다른 코드이다. 단지 물리적으로 코드를 배정했다는 차원에서는 비슷하지만 언어학, 디자인, 공학, 서체 등등 제반 학문적인 원리를 바탕으로 코드를 배정하였기 때문에 이러한 원리를 이해해야만 한글 처리 원리를 이해할 수 있다. 유니코드에서 사용한 원리는 매우 많고 다양하지만 이해를 돕기 위해서 필요한 요소를 중심으로 설명하므로 자세한 것은 참고 자료을 이용하기 바란다.

먼저 디자인 원리에서 16비트 크기, 의미론, 논리적 순서, 단일화, 동등 연속 등이 적용되었다.

- 16비트 크기라는 것은 말뜻 그대로 한 문자를 표현하기 위해서 16비트(2바이트) 크기로 사용한다.

- 의미적인 지식을 요구하는 파싱(parsing), 정렬, 기타 알고리즘에서 사용하는 의미론적 지식 수용한다.

- 글쓰기 방향과 관계된 논리적 순서는 왼쪽에서 오른쪽, 오른쪽에서 왼쪽으로, 위쪽에서 아래쪽으로 사용할 때 별도의 해석정보를 줄 수 있다.

- 단일화는 같은 문자를 한 번만 지정하는 원리로 한국·중국·일본·대만 등에서 주로 사용하는 한자는 형태는 같을지라도 음이 다르거나 각각 처리하는 방식이 다르기 때문에 통일시키는 것이다. 그런데 이것은 한국은 물론 다른 국가에서도 문제가 되는 부분으로 KS완성형 코드 제정에서는 논란이 되었던 '樂'자의 경우 한국어에서 한글 음절로 표현할 때는 '락', '악', '요'로 되기 때문에 이것을 구별하려면 별도의 원리를 만들어야 한다.

- 동등 연속(Equavalent Sequence)은 유니완성형 '각(0xac01)'자는 유니조합형으로 풀어쓴 'ㄱ(0x1100)' + 'ㅏ(0x1161)' + 'ㄱ(0x11a8)'과 같게 해석할 수 있다는 것인데, 이것으로 인하여 유니조합형으로 풀어쓴 것은 유니완성형과 섞여 있을 경우에 별도의 작업을 통하여 정렬이나 탐색에서 수행해야 한다.

코드 배정은 크게 8영역으로 나누어졌는데, {일반 스크립 영역, 심볼 영역, CJK 음성 및 심볼 영역, CJK 한자 영역, 한글 유니완성형 영역, 대용 영역, 사적인 사용 영역, 호환 및 특별 영역}이며, 이 중에서 한글과 직접 관련된 것은 유니조합형 자모를 처리하는 [일반 스크립 영역], 한자를 처리하는 [CJK 한자 영역], 한글 완성된 글자를 처리하는 [한글 유니완성형 영역] 등이다.

유니코드에서 눈에 띄는 특수 문자는 바이트 순서 문자(Byte Order Mark)가 있다. 바이트 순서 문자는 하드웨어마다 기록하는 방식이 다르기 때문에 발생하는 문제를 해결하기 위해서 사용하는 것이다. 예를 들어, 인텔용으로 저장한 비문자형 (int, long) 변수는 Sun, Hp에서는 바이트가 바뀌어 저장되므로 각각 해석을 다르게 해야 한다. 이러한 문제를 해결하기 위해서 바이트 순서 문자로 "0xfe, 0xff"를 연속적으로 맨 앞에 붙여서 기록한다. 이렇게 기록된 것은 인텔에서는 "0xff, 0xfe"로 읽히기 때문에 다음에 오는 모든 유니코드는 바이트 순서를 바꾸어서 해석하게 된다.

[예. 바이트 순서 해석 방법]

0xff, 0xfe -> 인텔용

0xfe, 0xff -> Sun, Hp

다음으로는 한글 처리 원리를 알아보겠다. 유니코드에서는 한글을 완성형(유니완성형)과 조합형(유니조합형) 두 가지 모두 처리할 수 있도록 되어 있는데, 현대 한글 11172자는 완성형처럼 미리 조합했다는 뜻으로 완성형(유니완성형)이라고 하고 초성 90자, 중성 66자, 종성 82자, 채움 2자 등 모두 240자의 자모로 조합할 수 있도록 했기 때문에 조합형(유니조합형)이라고 부른다.

유니완성형 코드는 KSC5601완성형 코드와는 다르게 일정한 조합 원리를 갖고 있어서 사용하기 편리하므로 쉽게 자소 정보를 뽑을 수가 있다. 그래서 유니완성형 코드는 유니조합형 코드로 변환이 가능하며, 역으로 유니조합형은 유니완성형으로 변환이 가능하다. 이 때, 유니완성형을 유니조합형으로 풀어 쓴 것을 해체한 코드라고 부르며, 역으로 유니조합형을 유니완성형으로 변환하여 쓴 것은 합성한(조합한 의미가 정확하지만 조합형이라는 말과 구분하기 위해서) 코드라고도 부른다. 유니조합형 코드는 초성, 중성, 종성 글자를 풀어서 사용하는 것으로 기존의 상용 조합형 코드에서 조합하는 원리와 똑같다. 단지 차이가 있다면 상용 조합형은 초성, 중성, 종성 각각 5비트씩 사용해서 2바이트를 유지하는 반면에, 글자 수만큼 크기를 2바이트씩 차지하므로 길이가 가변적이라는 것이다. 그런데 여기서 매우 중요한 점이 있다. 유니완성형 코드와 유니조합형 코드는 동일한 코드로 처리할 수 있다는 것이다. 즉, 유니완성형 '각(0xac01)'자는 유니조합형으로 해체한 'ㄱ(0x1100)' + 'ㅏ(0x1161)' + 'ㄱ(0x11a8)'자와 똑같은 글자로 처리해야 한다는 뜻이다. 이렇게 현대어 한글 한 글자를 표현하는데 두 가지로 표현하게 되므로 문제가 발생한다. 같은 글자를 두 가지로 표현하다 보니까 정렬이나 탐색에서는 별도의 조작을 수행해야만 가능하며, 옛한글이랑 섞여 있는 경우에는 더욱 복잡한 조작을 수행해야 하는 부담이 따른다. 만약에 유니코드 상태에서 유니조합형 코드와 유니완성형 코드를 섞어서 일반적인 정렬 프로그램을 사용하면 한글 순서가 뒤죽박죽 뒤엉키게 된다. 이와 비슷한 다른 예로써 'ㅘ'라는 글자도 'ㅗ' + 'ㅏ'로 해체하여 사용할 수 있으므로 더욱 복잡한 연산을 수행해야 한다. 그러면 왜 유니조합형과 유니완성형 두 코드를 동시에 수용했는지 궁금하지 않을 수 없다. 그 이유를 필자가 개인적으로 생각해보면 처리의 효율성을 높이기 위해서 결정한 듯하다. 현대어만을 다루기 위해서는 유니완성형만을 가지고도 충분하지만 현대어가 아닌 옛한글을 모두 처리하려면 이론적으로 대략 50만자가 된다. 이렇게 많은 자를 16비트 체계에서 처리한다는 것은 거의 불가능하므로 현대어처럼 유니완성형 형태로 유지하지 않고 자소별로 풀어서 240자를 배정한 것이다. 간단히 말해서 옛한글을 처리할 때와 미완성 음절 문자는 어쩔 수 없이 유니조합형을 사용해야 한다는 뜻이다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 유니코드에서 한자는 어떻게 되었는가

유니코드에서 한자는 앞서 말한 대로 한국·중국·일본·대만 등에서 주로 사용하기 때문에 각 국가별로 사용되는 한자를 부수와 획수에 맞추어 배열하였다. 그런데 이러한 문자 중에서는 각국에서만 독특하게 사용하는 경우가 있으며, 어떤 문자는 두 가지 이상을 표현하는 경우도 있고, 한국에서는 사용하지 않고 중국이나 일본에서 사용하는 문자에 대하여 한글 음절을 부여하는 문제도 발생한다. 이러한 문제는 유니코드에서 확실하게 해결하지 못 하였기 때문에 **한자**를 사용하는 국가끼리 협의하도록 되어 있다. 또한 국내에서는 한자에 대한 한글 음절 순으로 정렬하는 테이블 갖고 있어야 하며, 하나의 글자로 되어 있는 '樂'자와 같은 경우에 한글 음으로 변환할 때는 각각 '락', '악', '요'자로 해석해야 하는 문제가 있다.

더욱이 유니코드에서 제정한 한자는 CJK영역에 20,902자를 배정하였지만 국제 한자 특별 위원회에 제안된 한자는 27486자가 있기 때문에 현실적으로 유니코드에서 제정한 한자는 표준으로 받아들이기에는 어려운 문제가 있다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 유니코드는 한글 및 한국어를 완벽하게 처리할 수 있는가

결론부터 말하자면 완벽하게 처리하지는 못 한다. 현대어 처리에서는 KS완성형보다는 나을 지도 모르지만 조합형 코드보다 못한 부분이 있으며, 옛한글 처리에서도 중요한 문제가 있다. 옛한글이라고 하는 것은 크게 두 가지로 구분하는데 훈민정음 창제 이후에 사용되었으나 현재에는 사용하지 않는 한글을 말하고, 또 하나는 구결이라고 하는 것이 있는데 옛날에 한자(한문)를 우리말로 읽을 때 한문의 낱말 또는 구절 사이에 들어가는 것을 구결이라고 한다. (이 글에서는 옛한글과 구결을 구분한다.)

먼저 유니완성형 영역에서 처리하는 현대어 음절 11172자에 문제가 있다. 즉, 현대어 음절은 11172자가 아니라 정확하게 12319자이다. 이렇게 차이가 나는 것은 미완성 음절 문자 1147자가 있기 때문이다. 이것은 과거 KS완성형과 대립하는 과정에서 조합형에서 사용하는 완성된 음절을 이루는 글자가 11172자로 보았기 때문에 차이가 발생했다. 미완성 음절 문자는 초성:채움:채움(衁), 채움:중성:채움(葡), 채움:채움:종성(葖), 채움:중성:종성(葶), 초성:채움:종성 등의 글자 형태이다. 이러한 글자들이 현대어에서 빠지게 되면서 옛한글처럼 유니조합형에서만 처리할 수 있도록 배정되는데, 이것으로 인하여 앞으로(1995년 3월 이후) 개발될 운영체제에서는 현대어를 11172자만을 처리하게끔 유도한다. 이렇게 된 근본적인 원인을 살펴보면, 유니코드 2.0이 제정되기 전까지 KS완성형과의 논쟁에 집착한 나머지 조합형에서 현대어 음절이 정확하게 몇 자인지 모르고 있었기 때문이다. 실로 어처구니없는 일이지만 이미 "엎질러진 물"처럼, 엄연한 현대어이면서 옛한글을 처리하기 위해서 만들어 놓은 유니조합형 영역에서 함께 처리되는 결과를 낳는다. 게다가 이러한 잘못된 판단으로 유니조합형에서는 현대어 자모와 옛한글 자모가 뒤엉키게 되어 정렬이나 탐색에서 심각한 문제를 불러일으킨다.[그림. 유니조합형 코드를 보면 순서 뒤죽박죽이다.]

다음으로 옛한글을 살펴보자. 옛한글을 표현하기 위해서는 유니조합형 자모 코드 240자(초성 90자, 중성 66자, 종성 82자, 채움 2자)를 사용하면 이론적으로 약 50만 자의 음절을 표현할 수 있다. 그런데, 유니조합형 240자만으로는 옛한글을 충분히 표현할 수 없으며, 실제로는 104자(초성 124자, 중성 84자, 종성 133자)가 더 추가된 344자가 있어야지만 완벽하게 처리할 수 있다. 이러한 옛한글 자모를 조합하면 대략 139만 자를 표현할 수 있지만 실제로 문헌에 나타난 옛한글 음절은 대략 5,000자 정도밖에 안 된다는 점에 주의하기 바란다. 잠시 화제를 바꾸어서, 유니조합형(유니코드2.0)에서 104자의 자모 코드가 빠지게 되는 과정을 알아보자. 초창기 유니코드 1.x을 제정하던 시기에 한국에서는 유니코드에 대한 준비가 거의 없었으며, 게다가 그 당시에 국내에서는 표준 코드였던 KS완성형 코드를 조합형 코드로 개정하려는 데 관심이 많았던 때였다. 그러므로 옛한글 음절이나 자모에 대한 연구는 상당히 초보적인 수준이었는데 그 때까지 연구되어 발견된 자모가 대략 200자 정도였다. 이러한 상황에서 유니코드1.x가 제정되었는데, 이 때 조합형 코드로 채택된 자모 코드는 240자였다. 그런데 유니코드 1.x이 채택되자마자 옛한글 음절이나 자모를 본격적으로 연구하는 과정에서 새로운 음절과 자모를 발견하게 된다. 하지만 유니코드2.0에서는 새로 발견된 옛한글 자모를 수용하지 않았고 유니코드 1.0에서 채택된 내용만을 수용하면서 앞에서 지적한 104자의 자모가 빠지게 되었다. 여기서 한 가지 주의할 점이 있다. 유니코드 2.0이 제정된 이후에도 계속해서 옛한글 자모가 계속해서 발견되고 있으므로 유니코드 입장에서는 새로운 옛한글을 무조건 수용할 수 없었던 점도 이해해야 한다. 다시 말해서 국제 표준화 작업에 대한 충분한 대비를 하지 않고 있다가 "소 잃고 외양간 고치는 격"으로 뒤늦은 연구 작업을 하였기 때문에 우리의 잘못이 크다고 볼 수 있다. 또 다른 한 편으로는 짚고 넘어가야 할 중요한 점이 있다. 여러 가지 사정으로 인하여 유니코드1.x에서는 순서가 뒤엉킨 자모 배열를 유지하였지만 유니코드2.0에서도 이것을 고치지 않고 그대로 수용한 점은 이해하기 어려운 부분이다. 결국, 한국어에서 자모 배열을 무시한 코드가 또 다시 등장한 셈이다.

끝으로 구결의 경우를 살펴보자. 현재 발견된 구결 문자는 276자 정도로 보고 있으며, 한자에서 모양을 빌려왔기 때문에 언뜻 보아서는 다른 문자(한자, 그림 문자, 기호 문자)를 변형시키면 어느 정도 출력은 가능하다. 그래서 구결에 관계하지 않는 사람 입장에서는 이런 정도는 빠져도 괜찮지 않을까 하는 생각을 가질 것이며, 적은 수의 국어학자들이 알아서 적당히 해결해야 하는 상황이 되었다.

앞서 살펴본 것처럼 유니코드는 한글을 완벽하게 처리할 수 없지만, 과거 KS완성형이 지녔던 문제와 비교해보면 거의 무시할 수 있을 정도이다. 하지만 필자의 개인적인 생각으로 옛한글과 구결 문자는 빠지지 않았으면 한다. 우리가 과거 조합형을 포기하고 KS완성형을 채택한 후에 얼마나 많은 사람이 고통을 받았는지를 생각해 보면 앞으로 어떻게 해야 하는지를 알 수 있을 것이다. 문화 유산이라는 것은 한 번 포기하고 버리기는 쉬어도 되살려 복원하는 데에는 너무나 어렵고 힘든 일이기 때문이다. 지금은 옛한글과 구결이라는 것이 몇몇 학자들만의 문제이지만 먼 훗날 코드가 없기 때문에 사라져 버린다면 너무나 애석한 일이므로, 비록 옛한글과는 관계가 없더라도 많은 독자들이 관심이 가졌으면 한다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 한글 윈도우98과 문서처리기에서 한글 코드 및 한글 처리

컴퓨터에서 가장 중요한 것은 운영체제이고, 널리 사용되는 한글 윈도우 98에서는 KSC 5700 표준 한글 코드(유니코드)를 지원함으로써 현대어 한글 11,172자를 모두 표현할 수 있다. 여기서 주의할 점은 현대어 중에서도 미리 조합된 유니완성형만을 표현할 수 있다는 점이며, 또한 기존의 KS완성형과는 다르게 초성, 중성, 종성 자소 정보를 해석할 수 있다는 점이 커다란 차이점이다. 다시 말해서 한글 조합 원리를 반영한다는 뜻으로 유니완성형으로는 한국어 정보 처리에서 중요한 형태소 분석을 할 수 있다는 점이 중요하다. 그러나 현대어 한글 음절은 11172자가 아니라는 점에 유의해야 한다. 정확하게 말하자면 현대어 한글은 12391자이기 때문에, 윈도우98에서는 현대어 미완성 음절 문자 1147자와 옛한글을 표현할 수 있는 유니조합형은 처리할 수 없다.

그럼 윈도우98에서 실질적인 조합형(유니조합형)을 지원하기 어려운 이유는 무엇일까 궁금하다. 아마도 필자의 개인적인 생각으로는 240자모를 조합할 경우에 50만자에 가까운 음절이 필요하므로 현실적으로는 모두 처리하기 어렵다. 또 한편으로는 유니완성형과 호환성을 유지하도록 되어 있는 유니조합형 글자를 사용하면 정렬이나 탐색에서 추가 작업을 해야 하는 어려움이 있으며, 데이터 용량도 2~3배 정도 늘어나는 문제가 있다. 그래서 현재 수용 가능한 11172자를 미리 조합된 유니완성형 형태로 지원하는 듯하다. 그리고 유니완성형만으로도 과거 상용조합형 코드가 자랑처럼 여기던 현대어를 모두 포함하는 동시에 초성, 중성, 종성 음소 단위의 정보를 제공할 수 있기 때문에 커다란 문제가 없기 때문인 듯하다. 그런데 필자가 여기서 관심을 두는 것은 현대어 중에서 미완성 음절 1147[초성(19) + 중성(21) + 종성(27) + (중성(21) ⁓ 종성(27)) + (초성(19) ⁓ 종성(27))]자이다. 이 글자들은 일반인은 사용하지 않고 국어 내지는 국어와 관계된 출판업에 종사하는 사람만 사용한다고도 볼 수 있다. 그러나 실제로는 일반인에게도 중요할 수도 있다. 한국어는 영어와 달라서 자료에서 음소 단위로 정보를 처리하고 싶을 때가 드물지 않게 발생한다. 예를 들어, 자료에서 종성 '葉뿐이다'를 '葉 따름이다'로 바꾸고 싶을 때 입·출력이 안 된다면 곤란한 것은 자명하다. 물론 이보다 더욱 중요한 사람들이 있다. 바로 한글 및 한국어 정보처리를 담당하는 사람들이다. 이들이 주로 하는 일들은 한국어를 형태소 분석하여 자동색인 시스템, 맞춤법 검사기, 정보검색 시스템, 기계번역 시스템 등등에서 미완성 음절 문자 1147자를 사용하기 때문이다. 미완성 음절 문자 1147자는 단순히 입력과 출력만 하는 것이 아니다. 화제를 잠시 바꾸자면, 현재 산업 현장과 학계에서는 KS완성형 코드가 표준이던 시절부터 한글 및 한국어 정보처리를 위해서 정보를 충분히 표현할 수 있는 조합형 코드 내지는 자소코드를 각자 만들어 사용해 왔으며, 그것을 가지고 만든 형태소 전자사전을 기반으로 자동색인, 정보검색, 맞춤법 검사, 기계번역 등등에서 활용하고 있다. 그런데 이렇게 각각 다른 조합형 코드나 자소코드를 가지고 형태소 전자사전을 비롯하여 정보 데이터베이스를 개발함으로써 범국가적인 정보 데이터베이스 사업을 추진하는 과정에서도 코드에 대한 호환성과 표준화 문제가 발생하므로 상당한 지연과 불편함을 초래해 왔다. 전자사전에서 사용하는 데이터는 코드상의 문제로 인하여 내부적으로 정의해서 사용해 온 것이므로 정상적인 방법으로는 입·출력을 할 수 없어서 항상 변환과정을 거쳐야 하는 문제, 제작하는 사람마다 사용했던 코드값이 달랐기 때문에 실질적인 표준코드를 정해야 하는 문제, 설령 정상적인 표준코드(유니조합형)로 개발되었어도 미완성 음절 문자는 입·출력이 불가능한 문제 등등이 발생한다. 그러한 상황에서 실질적인 표준으로 사용할 수 있는 코드를 지원하는 운영쳬제가 보급된다면 아마도 그것을 가지고 정보처리용 자소코드에 대한 표준화가 급속하게 진전될 것이며, 그것이 범국가적인 정보 구축 사업에서는 비약적인 발전을 가져올 것이다. 필자가 미완성 음절 문자 1147자를 진정으로 중요하게 생각하는 이유가 바로 여기에 있다. 다시 윈도우98로 돌아가서, 유니조합형 코드는 이론적으로 50만 자까지 조합이 가능하므로 현실적으로는 당장 수용하기 어렵다. 이로 인하게 함께 처리되지 못하는 미완성 음절 문자 1147자는 기술적으로도 시간적으로도 그다지 어려운 일이 아니었음에도 유니코드2.0에서 현대어 음절을 잘 못 배정하였기 때문에 1147자가 빠져 버렸다는 점은 필자를 비롯하여 한국어 정보처리와 관계된 종사자들에게 아쉬움을 남기는 부분이다. 한글 윈도우98의 입장을 심정적으로는 이해할지라도 윈도우 프로그램의 업그레이드 주기를 생각해 보면 빠져 있는 조합형(유니조합형) 코드가 하루라도 빨리 지원되었으면 하는 바램을 남겨 두고 다음으로 한자의 사정을 알아보자.

윈도우98에서 한자를 살펴보면, 유니코드에서 제정한 한자는 CJK영역에 20,902자를 배정하였다. 그러므로 유니코드를 지원하는 윈도우98에서도 약 20000자의 한자를 지원한다고 생각할 수도 있지만, 실제로는 7744자만을 지원한다. 이것 역시 나름대로 이유가 있다. 유니코드에서 제정한 한자는 한국, 중국, 일본, 대만 등 한자를 사용하는 국가의 것을 배정한 것으로, 이 중에는 우리 나라에서 사용하지 않는 한자도 있으며, 다른 한 편으로는 국제 한자 특별 위원회에 제안된 한자는 27486자가 있기 때문에 현실적으로 유니코드에서 제정한 한자는 표준으로 받아들이기에는 어려운 문제가 있다. 이와 같이 현실적으로 사용하기 어려운 상황에서 한자를 처리하기 위해서는 기존의 표준코드(KSC5601-1987 완성형[4888자]과 KSC5657-1991 확장 표준 코드[2856자])에서 사용했던 7744자를 현실적으로 수용하는 범위내에서 유니코드와 호환성이 유지되는 한자를 처리할 수밖에 없다. 아마도 이러한 이유 때문에 윈도우98에서는 7744자의 한자를 지원하는 듯하다.

앞에서 검토한 한자와 옛한글에 대한 보완책으로 유니코드를 지원하는 응용 프로그램을 생각해 볼 수 있을 것이다. 현재 응용 프로그램 중에 가장 널리 사용되는 문서처리기에서는 유니코드를 완벽하게 지원하는 프로그램은 없으며 단순하게 현대어 한글 유니완성형 11,172자만을 지원한다. 그래서 옛한글과 미완성 음절 문자 1147자는 유니코드를 사용하지 않고 내부적으로만 조합형 코드를 이용하여 부분적으로 지원하고 있는 문서처리기가 있다. 이 부분에서 주목할 필요가 있다. 과거에는 KS완성형이 표준이었기 때문에 상대적으로 조합형 방식을 채택한 문서처리기들은 완벽한 한글 처리 능력을 자랑해 왔다. 그러면 과연 완벽한 한글 처리에 대하여 다시 한번 검토해 보자. 과거에는 현대어 음절을 충분히 표현할 수 있느냐의 문제에 초점을 맞추었으나 이제는 상황이 달라졌다. 이제 현대어는 시스템 차원에서 충분히 처리할 수 있으므로 옛한글 처리가 중요한 문제로 부각되고 있다. 국어학자 입장에서 옛한글이라고 말할 때는 한글 형태와 그렇지 않은 구결도 포함하지만 일반인이 현실적으로 이해할 수 있는 한글 형태만을 가지고 검토해 보겠다. 현재 옛한글을 처리할 수 있는 프로그램은 각각 내부적으로 특수 처리 방법을 사용하고 있으므로 여기서 사용하는 옛한글은 입·출력만 가능하다. 그래서 데이터로서는 접근할 수 없는 죽어 있는 정보에 가깝다. 좀더 이해하기 쉽게 말하자면 화면과 종이에서만 사용할 수 있고, 데이터베이스와 같은 정보 구축이나 정보검색에서는 현실적으로 사용하지 못하고 있다. 물론 이에 대한 주원인은 표준적인 코드가 아니기 때문이겠지만 이들 문서처리기에서 사용하는 데이터를 다른 분야에서 사용할 수 있도록 공개적이지 못한 점도 검토해보자. 돌이켜 보면, 과거 KS완성형 코드가 표준이었을 때도 상용조합형은 표준 코드가 아니었음에도 불구하고 데이터 형식이나 기타 정보를 공개적으로 사용했었다. 이와 마찬가지로 생각해보면 옛한글과 문서처리기에서 사용하는 한글에 관련된 다른 정보(확장 한자, 구결)에 대하여 공개적이지 못한 태도는 배려의 문제라고 볼 수 있다. 여기서 공개적이라는 말은 개발에 관련된 특수한 비법이나 회사에 치명적인 손상을 입힐 만한 정보를 말하는 것이 아니다. 이들 프로그램에서 사용하는 데이터를 보다 쉽게 개방적으로 접근할 수 있게끔 하는 공개적인 태도를 말한다. 비록 이들 회사는 공개할 어떠한 의무나 책임이 없다. 하지만 이들이 사용하는 정보는 실로 놀라우리만큼 중요한 국가적인 자산이므로, 이 정보를 사용하지 못한다면 눈에 보이지 않는 국가적인 손실이 엄청나기 때문이다. 즉, 우리의 자랑스러운 민족 문화를 정보화 사회로 이끌어 주는 다리와 같은 역할을 할 수 있기 때문에 중요하게 인식해야 한다. 필자 개인적인 생각으로는 적극적인 배려의 자세가 부족한 것은 아닌지 의문을 던져 보고, 다른 한편으로는 보다 넓은 마음으로 생각해 주었으면 하는 바램을 갖는다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 한글 표준 코드는 어떻게 처리되어야 하는가

앞서 KS완성형의 문제점을 설명하면서 한글 코드가 지녀야 하는 기본적인 원칙을 설명하였다. 여기서는 현재와 앞으로의 한글 코드는 어떠한 모습을 갖추어야 하는지 살펴보자.

1. 한글 원리가 반영해야 한다는 점이다. 표현 가능한 문자는 모두 처리할 수 있어야 하며, 옛한글도 현대어처럼 다룰 수 있어야 하고, 비한글 문자(한자, 영문자, 일문자, 기타 문자)도 사용하는 데 문제가 없어야 한다.

2. 한국어 정보 처리를 위해서는 음절을 넘어서 음소 단위로 처리할 수 있어야 한다.

3. 다른 프로그램과 충돌이 없어야 한다.

4. 처리된 자료 및 기술은 산업 현장, 학계, 정부 기관에서 공유하도록 공개적이어야 한다.

5. 국제적인 표준 규격을 따라야 한다.

6. 정부의 정책적인 지원이 있어야 한다.

이 중에서도 가장 중요한 것은 뭐니뭐니 해도 한글 원리를 충실하게 반영해야 한다. 지금까지 실질적인 표준 한글 코드였던 KS완성형은 이러한 원칙을 무시하였기에 심각한 문제를 앉고 있었다. 그런데 KSC5700(유니코드2.0)은 이러한 문제를 해결할 수 있으므로 매우 바람직하다고 생각한다. 유니코드는 처리 원리상 현대어 11172자는 물론 미완성 음절 문자와 옛한글을 처리할 수 있도록 되어 있지만 실질적으로 처리되려면 어느 정도 시간이 필요할 듯하다. 이와 더불어 음소 단위로 처리할 수 있어야 하는데 유니코드는 이 부분을 완벽하게 소화할 수 있어서 기대되는 바가 크다.

다음으로는 주목해야 할 점은 산업 현장에서 공유하도록 공개적이어야 한다는 부분이다. 이에 대한 것은 구체적인 사례를 통하여 함께 생각해보자. 지난 1995년 마이크로소프트사가 한글 윈도우95를 발매하는 과정에서 확장 완성형 코드를 발표함으로써 많은 논란을 불러 일으켰다. 그 당시 관련 업계는 물론이고 언론에서도 크게 부각시켰던 것 중에 하나는 발표되지 직전까지 확장 완성형 코드에 대한 언급이 없었다는 점이다. 업계와 언론에서 비난의 화살이 쏟아지자 이미 프로그램 개발을 완료한 시점에서 확장 완성형 코드를 포기하기 어렵다는 입장을 취하게 되고, 그것이 마이크로소프트사의 계획적인 음모로까지 비추어지게 되었다. 사실상 업계에서 시장을 주도하고 있는 회사가 새로운 표준으로 자리잡을 수도 있는 문제를 일방적으로 발표함으로써 이에 대한 반발은 물론이고 국가적으로 중요한 문제였으므로 공개적으로 공청회를 거쳐야 한다는 여론이 들끓었다. 결국 이것은 마이크로소프트사의 회사 이미지를 실추시키는 결과가 되었다. 필자 개인적인 생각으로는 확장 완성형 코드가 음모나 횡포의 결과이기보다는 그 동안 처리할 수 없었던 8822자를 KS완성형 코드와 호환성을 유지하는 범위 안에서 처리할 수 있도록 하는 것이 주목적이었으며, 마이크로소프트사도 확장 완성형 코드를 채택하는 과정에서 내부적으로는 오랫동안 많은 토론과 고민을 한 것으로 전해 들었으며, 발표하기 직전까지 국내 개발 업체와 어느 정도 협의를 해왔었다. 더욱이 상당히 오랜 기간 동안 베타 테스트를 거쳐 왔기 때문에 음모적으로 발표한 것과는 거리가 멀었다는 점이다. 하지만 직접적으로 관련이 적은 산업 현장, 학계, 정부 기관 등등 기타 관계자들과 별다른 협의나 공개적인 토론회도 없이 발표됨으로써 독점기업 의도적인 음모와 외국 회사의 일방적인 횡포로 비추어지게 되었다. 바로 이 부분이 우리에게 중요한 점을 시사한다. 국가적으로 중대한 영향력을 미칠 수 있는 한글 코드를 개정하거나 채택하는 데 있어서 산업 현장을 비롯하여 학계와 정부 기관에서 함께 고민하고 수용할 있도록 공개적으로 검증을 받으면서 전개되어야 한다는 교훈을 안겨 준 셈이다. 또 다른 사례로는 현대어는 물론 옛한글과 한자를 훌륭하게 지원하는 프로그램을 살펴보자. 이들 프로그램에서 옛한글과 확장한자(KS완성형에 없는 한자)는 입·출력을 할 수 있지만 정보로서는 사용할 수 없는 죽어 있는 데이터라는 점을 앞에서 밝혔다. 앞에서 지적했듯이 이들 프로그램이 처리하는 정보(현대어, 옛한글, 한자)는 값어치로 따지기 어려우리만큼 귀중한 자산이지만 공개적으로 자료를 접근할 수가 없어서 안타까움을 금할 수 없다. 즉, 보이지 않는 국가 재산을 썩히고 있는 셈이다. 이러한 정보를 쉽게 접근할 수 있도록 배려한다면 한국어 정보 데이터베이스 구축이라는 국가적인 사업에서 획기적인 발전을 가져왔을 텐데 하는 아쉬움이 남는다. 비록 이들 프로그램을 만드는 회사는 어떠한 책임이나 의무가 없지만 국가적인 발전을 위해서 보다 넓은 마음으로 배려하였으면 한다.

끝으로 정부의 정책적인 지원을 살펴보자. 한글 코드 문제는 일반 상품과는 달리 사용하는 과정에서 문제점이 있거나 그것으로 인하여 다른 많은 사람에게 영향을 미칠 경우에 개개인이 나서서 해결하기가 어렵다. 특히 기업이나 표준 규약을 상대로 개인이 문제를 해결하기란 계란으로 바위를 치는 것보다 더욱 어려운 문제이다. 따라서 정부는 대한민국 국민을 대신하여 한글 코드 문제를 적극적으로 다루었으면 한다. 왜냐하면 정보화 사회에서 정보고속도로가 대동맥이라면 한글 코드는 영양분을 실어 나르는 피와 같은 구실을 하기 때문이다. 이렇게 생명처럼 소중한 한글 표준 코드에 문제가 있다면 문화의 꽃과 같은 언어를 제대로 사용할 수 없다는 것과 마찬가지이므로 정부가 앞장서서 정책적인 지도와 노력을 기울여야 하는 것은 너무나 당연하다. 흔히 잘 못 끼워진 첫 단추로 비유되던 KS완성형 코드 제정 당시에 좀더 적극적으로 문제의 본질을 파악하고 대처했더라면 지금까지 우리가 해 왔던 소모적인 논쟁은 피할 수도 있었을 텐데 하는 아쉬움을 갖는 사람이 많다. 또한 앞에서 지적했듯이 한글 코드에서 현대어는 물론 옛한글과 한자는 자랑스런 우리의 민족 문화 유산을 계승하고 발전시키는 데 중요한 역할을 하고 있음에도 불구하고 제대로 처리하지 못하고 있다. 설령 응용 프로그램 단계에서 이러한 문제를 해결한다고 하더라도 공개적으로 정보를 공유하지 못하므로 국가적으로 커다란 손실을 입고 있다. 만약 정부가 이러한 문제에 적극적으로 나서서 자원을 공유하도록 유도할 수 있는 정책을 개발하고 지원한다면 보다 나아질 수도 있을 것이다. 왜냐하면 영리를 추구하는 기업이 별다른 보상도 없이 자신만의 독특한 기술로 일구어 낸 정보를 공개한다는 것은 한 마디로 생존을 포기하는 것과 마찬가지이기 때문에 현실적으로는 거의 불가능한 일이다. 막연히 국가적인 이익을 위해서 공유하자는 것보다는 국가 발전에 기여하는 정도에 따라서 보상을 받을 수 있는 정책을 개발하고 유도한다면 현실적으로 풀어 갈 수 있으리라고 본다. 이와 같이 기업과 사용자 모두가 함께 발전할 수 있게끔 하는 것이 정부의 역할이라고 생각한다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 코드 변환에 대하여

코드 변환은 크게 세 가지 상태를 변환하는 것이다. 앞서 말한 대로 여러 종류의 코드가 있지만 실질적으로 사용하는 코드는 KS완성형, 조합형(상용, 표준)과 유니코드밖에 없기 때문에 각각의 변환 과정에 대하여 소개하겠다. 이 글에서는 조합형은 상용 조합형을 다루며, KS완성형은 윈도우95 이후에 사용되는 확장 완성형을 수용하며, 유니코드는 다른 코드와 호환이 가능한 문자 1바이트 아스키 문자와 온전하게 완성된 현대어 11,172자를 중심으로 소개한다. 유니코드에서 사용하는 한자는 20,902자이지만 앞에서 말한 것처럼 표준으로 정착되기 어려운 점과 다른 코드와 호환되지 않는 문자가 있으므로 제외하고, 유니조합형으로 부르는 자모 코드는 옛한글을 표현하지 못하므로 제외하고, 그밖의 비한글 문자 중에서 KS완성형과 호환되도록 배열된 코드가 있지만 이 글에서는 제외한다. 즉, 유니코드에서는 다른 코드와 호환하는데 문제가 있는 부분은 제외하기 때문에 1바이트 아스키 문자와 온전하게 완성한 현대어 한글 11,172자만 변환한다. 또한 이 글에서 기본적인 원리를 바탕으로 설명하는 대신에 구체적으로 변환하는 과정은 함께 제공되는 소스 코드를 참고하기 바란다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● KS완성형 <-> 조합형(상용)

이것은 본지를 비롯하여 다른 많은 서적에서 소개되기도 한 부분이지만, 윈도우 95이후에 사용되는 확장 완성형 한글 문자(줄여서 확장 한글이라고)에 대한 소개가 대부분 빠져 있으므로 간단하게 보충하여 설명하겠다. KS완성형은 다른 코드와 다르게 기본적인 형태소 해석 정보를 갖고 있지 않기 때문에 테이블 방식으로 다루어야 한다. 좀 더 쉽게 말하자면, 각 글자에 대응하는 꼬리표를 사용해야 하는데, 꼬리표를 모르면 변환할 수 없다. 그래서 전체적인 꼬리표 내용을 하나의(혹은 여러 개의) 표로 기억해야 한다. KS완성형은 크게 1바이트 문자와 2바이트 문자로 구분하며, 2바이트 문자는 한글 영역(기본 한글), 확장 한글, 한자 영역, 전자로 된 1바이트 문자 호환 영역, 다른 국가 문자 영역(일본어 문자, 러시아 문자, 그리스 문자 등등), 기타 기호 문자 영역으로 구분하면 이해하기 쉽다.

|  |  |  |  |
| --- | --- | --- | --- |
| KS완성형 영역 | KS완성형 범위 | 조합형 범위 | 특징 |
| 1바이트 아스키 문자 | 0x20 - 0x7f | 0x20 - 0x7f | 둘 다 똑 같다. |
| 기본 한글 문자(2350자) | 0xb0a1 - 0xc8fe 가: 0xb0a1 각: 0xb0a2 간: 0xb0a2 ...: ... 힝: 0xc8fe | 0x8861 - 0xd3b7 가: 0x8861 각: 0x8862 간: 0x8865 ...: ... 펽: 0xd3b7 | KS완성형의 글자 표로 배열되어 있기 때문에 무조건 표를 만들어 변환해야 한다. |
| 확장 한글(8822자) | 衣: 0x8141 펽: 0xc65a | 衣: 0x8863 펽: 0xd3bd |
| 한자 (4888자) | 伽: 0xcaa1 詰: 0xfdfe | 伽: 0xe031 詰: 0xf9fe |
| 전자로 된 1바이트 호환 문자 | 、: 0xa1a2 я: 0xacf1  | 、: 0xd9a2 я: 0xdef1  |
| 다른 국가 문자 |
| 기타 기호 및 선문자 |
|  |

필자가 처리하는 방법말고도 다른 방법을 사용하는 책들이 있으므로 관심있는 독자들은 참고 자료들을 보면 되는데, 한 가지 주의할 점은 확장 완성형에 추가된 확장 한글 문자에 대한 부분이 없다는 점이다.[그림. 확장완성형 코드표]

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● KS완성형 <-> 유니코드

유니코드는 1바이트 문자와 호환되는 부분과 유니완성형 부분만을 변환한다. 유니코드를 제정하는 Unicode,inc.에서는 인터넷(www, ftp)을 비롯하여 책을 통하여 변환에 필요한 정보를 제공하고 있다. 특히 온전한 현대어 한글과 한자에 대해서는 다른 코드에 대응하는 코드값을 표기하였기 때문에 쉽게 이해할 수 있다. 여기서도 KS완성형 코드가 지닌 특성 때문에 테이블 방식으로 1:1로 변환해야 하고, 온전한 현대어 한글에 대한 대응 코드를 예로 보이겠다.

가: 0xb0a1 <-> 0xac00

각: 0xb0a2 <-> 0xac01

衣: 0x8141 <-> 0xac02 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

衤: 0x8142 <-> 0xac03 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

간: 0xb0a3 <-> 0xac04

...: ...

펺: 0xc64f <-> 0xd7a0 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

펻: 0xc650 <-> 0xd7a1 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

펼: 0xc651 <-> 0xd7a2 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

펽: 0xc652 <-> 0xd7a3 [확장 완성형 - 기본 완성형에는 대응 코드 없음]

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

● 조합형 <-> 유니코드

유니코드는 1바이트 아스키 문자와 온전한 현대어 한글인 유니 완성형 코드만 처리한다. 유니완성형 코드는 KS완성형 코드와는 근본적으로 다르게 형태소 정보를 사용한다. 그래서 조합형 코드처럼 초성-중성-종성 자소 정보를 쉽게 구할 수 있으며, 조합형 코드와 변화하는 과정도 매우 간단하다. 다만, 조합형에서 사용하는 미완성 음절 문자는 변환할 수 없다는 점에 유의하기 바라며, 먼저 초성-중성-종성 자소 정보를 구한 후에 각각의 구성 원리에 맞추어 합성하면 된다.

**▶ 조합형 코드에서 자소 구하기**

: 최상위비트(MSB)를 제외한 나머지를 차례대로 5비트씩 쪼개면 각각 초성, 중성, 종성 자소가 된다.

kssmcode(조합형 문자) = MSB(최상위 비트[1]) + ChoJaso(초성[5]) + JungJaso(중성[5]) + JongJaso(종성[5])

= (0x8000 | (ChoJaso << 10) | (JungJaso << 5) | (JongJaso))

ChoJaso = ((BYTE)((((HGCODE)(kssmcode)) >> 10) & 0x1f))

JungJaso = ((BYTE)((((HGCODE)(kssmcode)) >> 5) & 0x1f))

JongJaso = ((BYTE)(((HGCODE)(kssmcode)) & 0x1f))

**▶ 유니완성형에서 자소 구하기:**

유니완성형 문자: UncCode

BaseVal!(유니완성형 시작 위치) = 0xac00

ChoJamoNum(초성 문자 개수) = 19자

JungJamoNum(중성 문자 개수) = 21자

JongJamoNum(종성 문자 개수) = 28자(종성 채움 상태를 추가)

JungJongNum(중성과 종성의 합성 개수) = JungJamoNum \* JungJamoNum

UncCode = ((((ChoJaso \* ChoJamoNum) + JungJaso) \* JungJamoNum) + JongJaso + BaseVal!)

UncInx = UncCode - BaseVal!

ChoJaso = (UncInx / JungJongNum))

JungJaso = (UncInx % JungJongNum) / JongJamoNum

JongJaso = (UncInx % JongJamoNum)

..........................................................................................[*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

■ 정보는 자원이자 곧 국가 경쟁력이다.

아직도 인터넷이나 PC통신에서는 한글 코드 문제에 대한 노골적인 불쾌감이나 부정적인 시각이 많다. 경우에 따라서는 우리가 조합형을 사수하지 못했다는 점이 역사적으로 오점을 남기게 될 지도 모른다는 생각을 하기도 했었다. 그러나 그렇게 흥분한다고 해서 해결할 수 있는 문제가 아니었음을 차츰 깨닫게 되었다. 당시에는 한글 코드 문제가 이렇게까지 심각한 문제라고까지 받아들이기 어려운 상황이었으므로 누구를 탓하기도 어려울 듯하다. 오히려 우리 모두에게 더 큰 문제가 있을 수도 있다. 과거 KS완성형 코드가 제정될 때부터 문제를 푸는 과정에서, 해결보다는 논쟁거리로 몰아부치는 경향과 그 논쟁 자체에 집착한 것은 아닌지 반문하고 싶다. 앞에서 밝혔듯이 현대어에 대한 음절수를 정확하게 생각한 적이 없었기 때문에 유니코드 2.0에서조차 미완성 음절 문자가 현대어 영역에서 빠지게 되고 이로 인하여 현대어 자모와 옛한글 자모의 순서마저도 뒤엉키게 하는 결과를 가져왔다. 이렇게 잘못된 코드 배정은 다른 나라에서 한 것이 아니다. 우리가 직접 만들어서 제출한 것이다. 아마도 그 동안 코드 논쟁에 쏟아 부었던 노력에 3%만이라도 현대어 음절에 관한 연구를 하였더라면 이러한 결과를 자초하지는 않았을 것이다. 마치 "3%의 소금이 바다를 짜게 한다"는 말처럼, 생각을 제대로 하고 있던 몇 사람만이라도 있었더라면 똑같은 어리석음은 피할 수도 있을 텐데 하는 아쉬움이 남는다.

흔히 지금에 와서 결과만 보고 생각해 보면, 과거 첫 단추부터 잘 끼었더라면 하는 아쉬움이 맴돌 것이다. 하지만 "역사에서는 가정이란 없다"라는 말처럼 근·현대사에서 우리가 현명하게 대처했더라면 뼈아픈 식민지 지배를 받지 않았을 텐데라고 생각하는 것과 비슷한 문제처럼 다가온다. 한국 근·현대사처럼 한글 코드도 험난한 과정을 거쳐 지금까지 왔다. 어쩌면 역사로부터 우리가 배운 진정한 교훈이라면 앞으로는 이런 일들을 되풀이하지 않도록 현재에 최선을 다하는 자세일 것이다. 그렇게 하기 위해서는 지나온 한글 코드 변천 과정과 현재의 한글 코드 성격을 살펴볼 필요성이 있고, 그런 의미에서 한글 코드에서 가장 중요한 영향을 미칠 한글 윈도우98과 널리 사용되고 있는 프로그램이 지닌 성격과 의미를 살펴보았다.

이 글은 지나온 논쟁을 재현하기보다는 현재까지의 과정과 앞으로 발전시켜야 할 방향에 맞추어 썼다. 그래서 가능한 한 객관적인 상황과 현실적인 이유들을 되짚어 보려고 노력하였으며, 또한 앞으로 해결해야 할 과제들을 점검해 보았다. 이제 한글 코드는 단순히 코드 배열의 문제가 아니다. 정보가 자원이자 국력인 정보화사회에서 한글 코드는 국가 생존을 좌우할 핵심적인 무기가 될 지도 모른다. 특히 우리 나라 산업 현실을 고려할 때, 한글 윈도우98은 단순히 어느 한 기업의 상품이기 이전에 국가 경쟁력을 좌우하는 아주 중요한 매개물일 수도 있다. 그런 의미에서 한글 윈도우98에서 처음으로 유니코드(한국어 관련)를 적극적으로 수용함으로써 시스템 차원에서 제대로 된 한글을 사용할 수 있게 된 점은 높이 평가할 만하다. 우리가 첫 단추를 잘못 끼운 지도 벌써 10년이 넘었다. 10년이면 강산도 변한다는 말처럼 한글 코드도 많이 변해 왔고, 변해 오는 과정에서 소모적인 논쟁으로 많은 시간을 허비하기도 하였다. 결국 우여곡절 끝에 유니코드라는 범세계적인 표준 코드와 함께 KSC5700이라는 한글 표준 코드로 자리 잡혀가고 있다. 아마도 조만간에 유니코드라는 이름으로 한글 코드 문제는 십여 년간의 기나긴 싸움에서 대단원을 마칠 듯하며, 그 동안의 소모적인 논쟁은 결코 헛된 것이 아니었음을 강조하고 싶다. 그 과정에서 우리들의 열정과 노력을 통하여 일구어낸 경험이야말로 훌륭한 밑거름이 되었기 때문이다.

끝으로 유니코드2.0에서 한글 코드 배정을 위해서 노력해 주신 많은 관계자 여러분께 진심으로 고맙다는 감사의 마음을 전하며, "**한글 코드는 곧 국가 경쟁력**"이라는 점을 다시 한 번 강조하면서 이 글을 마친다.

............................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

\*\*\*\*\*\*\*\*\*\*\*\*\*\* 이 부분부터는 박스로 처리됩니다.

**▶ [박스] ISO-2022 규격**

이것은 아스키 부호 규격(ISO-464)의 7비트 부호를 확장하여 2바이트 부호 확장법을 정의한 것이다. 128자를 표현할 수 있는 아스키 부호 체계로는 많은 문자를 가지고 있는 동양권 국가에서는 사용할 수 없었다. 이러한 이유로 인하여 아스키 부호를 확장하여 2바이트 이상의 코드 체계를 사용할 수 있도록 만든 부호 체계 확장법을 ISO-2022라고 한다. ISO-2022의 구조를 살펴보면 다음 표와 그림과 같다.

ISO-2022에 의하여 확장 문자 영역으로 실제 사용 가능한 공간은 D영역으로 한정되어 있다. 2바이트 문자의 각 바이트의 최상위 바이트(MSB)는 모두 같아야 한다. 그러나 D영역에서 사용할 수 있는 문자수는 8836자의 글자만 사용할 수 있으므로 여러 나라의 문자를 한꺼번에 처리할 수 없다. 그래서 각 나라별로 문자 세트를 지정하여 사용한다.

---[그림 추가 - ISO-2022 구조]를 여기다 추가합니다. ---

**▶ [박스] Unicode와 ISO-10646의 관계**

ISO-10646은 국제 표준화 기구(ISO: International Standard Organization)와 국제 전자 기술 위원회(IEC: International Electrotechnical Commission)의 산하 위원회인 ISO/IEC JTC 1(Information Technology) SC 2 (Coded Character Sets) WG 2 (Multiple-Octet Codes)에서 제정 중이던 국제 표준이었다. 그에 비하여, Unicode는 Apple, Metaphor, RLG, Sun, Xerox 등이 1989년에 결성하여 만든 Unicide Working Group에서 컴퓨터에서 쉽게 사용할 수 있는 다국어 지원을 목적으로 제정하던 업계 표준이었다.

1991년 Unicide Working Group은 Unicode ,Inc.로 조직 체계를 바꾸고, 지지부진하던 ISO-10646의 다국어 부분을 Unicide사에서 제정하는 표준을 그대로 사용할 것을 국제 표준 기구에 제안하였다, ISO-10646 중에서 UCS-2 BMP부분은 Unicide와 동일하다.

JTC: Joint Technical Committee

SC: Subcommittee

WG: Working Group

**▶ [박스] KSC5700코드와 문제점 - 제 1차 한글 코드 표준화 위원회 회의록**

공업진흥청이 1995년 12월 7일 KS규격으로 제정 고시한 코드로 조합형으로 활용 가능한 한글 자모 240자와 완성형 방식으로 (완성형 코드와는 다른 의미) 사용 가능한 한글 11172자로 구성된 국제 문자 부호 체계이다. 이 규격은 완성형과 조합형을 동시에 수용할 수 있게 되어 있으며, 현대 한글 문자를 모두 표현할 수 있어서 KSC5601-1987완성형 코드가 지녔던 문제점을 해소할 수 있을 것이라고 한다. 이것은 유니코드2.0에서 배정된 한글 코드와 일치한다.

#. 한글과 한자의 정렬(Sorting) 문제

- 한자는 한국, 중국, 대만, 일본에서 사용하는 한자들을 획순에 의해서 배정하였기 때문에 한국어에서는 사전 순으로 변환해야 만 순서대로 처리할 수 있다. 또한 중국이나 일본에서만 사용하는 한자의 경우 새로운 음을 부여해야만 사용할 수 있으며, 폰트를 만들어야 사용할 수 있다.

- '樂'자와 같이 같은 글자이지만 음이 다른 경우에 어떻게 처리할 것인가.

- 옛한글은 조합 방식만으로 표현하기 때문에 별도의 정렬 프로그램을 개발해야 한다.

#. 옛한글 문제

- 아직 발견되지 못한 옛한글은 빠져 있기 때문에 새로운 옛한글이 발견될 경우 커다란 문제가 발생한다.

- 일반 사용자는 사용하지 않는 옛한글을 국어학 연구를 위해서 모두 추가하는 것은 효율적이지 못하다.

- Window 차기 버전에서는 완성형 11172자만 지원할 것이므로 대부분의 소프트웨어에서는 완성형만 지원하게 된다.

- 조합형을 위한 한글 자모 240자의 위치가 변하면 새로운 소프트웨어를 개발해야 한다.

- 한글에 대한 명확한 정의가 없어서 옛한글, 구결, 한자까지 한글로 보아야 하는가

**▶ [박스] 유니코드에서 현대 한글 완성형 11172자를 획득하는 과정**

유니코드는 1989년부터 제정되어 현재는 버전 2.1까지 발표되었으며, 현대어 한글 완성형 11172자를 배정받은 것은 1995년 버전 2.0에 이르서였다. 그 동안은 버전 1.0과 버전 1.1에서도 한글 코드에 대하여 논의는 있었는데, 현대어를 모두 표현할 수 있는 조합형 방식의 코드는 미국, 중국, 일본 등이 반대하여 배정받지 못 했다. 조합형 방식은 이론적으로 11172자를 표현하기 위해서 32768자(=2의 15승)를 수용할 수 있는 영역이 필요하다. 그런데 유니코드에서 표현 가능한 문자 65536자 중에서 절반을 한국이 사용하는 결과가 되므로 다른 국가의 심한 반발을 불러 일으켰다. 당시로는 코드 영역의 크기는 마치 영토의 문제처럼 대립하게 되었으므로 조합형 방식이 배제되고, 기존의 표준 코드인 KS완성형 위주로 배정받게 된다.

그러나 이렇게 됨으로써 한국은 물론 국제적으로도 유니코드를 사실상 쓸모없는 코드(더미코드[Dummy-code])로 전락시키는 결과를 가져온다. 이때 때마침 윈도우95와 윈도우NT에서의 한글 지원 문제로 고심하던 마이크로소프트사도 우리 나라의 문자 코드 전문 위원회의 활동에 참가하게 되고, 1995년 3월에 가서야 버전 2.0이 채택되면서 간소화시킨 온전한 현대어 한글 완성형 11172자를 확보하게 된다. 물론 이 때에도 중국 등의 거센 반대가 있었지만 미국 등 다른 대표들의 도움으로 표결에서 한국측 의견이 최종적으로 채택된다. 이로써 한국은 유니코드에서 두 번째로 많은 문자를 배정받는 국가가 된다. 가장 많이 문자를 배정한 영역은 한자이지만 엄밀한 의미에서는 중국식 한자가 아니고, 한국·중국·일본·대만 등에서 사용하는 거의 모든 한자를 부수와 획순으로 배정하였기 때문에 한국·중국·일본·대만 등의 한자이므로 실질적으로는 한국이 가장 많은 문자를 배정받는 국가인 셈이다.

**▶ [박스] 한글 코드별 효율성 비교 검토**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 완성형 | 조합형(상용, 표준) | 유니코드 |
| KS완성형 | 확장 완성형 | 유니완성형 | 유니조합형 |
| 한글 조합 원리와 음소 정보 | X | X | O | O | O |
| 현대어 음절 표현 | 2350자 | 11172자 | 11172자 | 11172자 | 11172자 |
| 미완성 음절 문자  | X | X | O | X | O |
| 정렬에서 사전 순서 유지 | X | X | O | O | X |
| 저장효율 | 2byte | 2byte | 2byte | 2byte | 2 \* n byte |
| 옛한글 표현 | X | X | X | X | O |
| 한자 표현 | 4888자(윈도우98에서는 7744자) | 약 20000자(규정상) |
| 형태소 분석에서 효율성 | 낮다 | 낮다 | 보통 | 보통 | 높다 |
| 운영체제 지원정도 | 매우 높다 | 낮다 | 없다 | 낮다 | 없다 |
|  |

완성형은 기본완성형(KS완성형)과 확장완성형으로 구분하고, 조합형은 상용조합형과 표준 조합형으로 구분하지만 거의 비슷하므로 하나로 다루며, 유니코드는 완성형 방식과 조합형 방식의 차이가 나므로 구분한다. 한글 조합 원리는 음절을 이루었을 경우에 초성, 중성, 종성에 대한 정보를 구할 수 있는 정도를 말하며, 현대어 음절 표현은 표현 가능한 음절수를 말한다. 미완성 음절 문자는 현대어 중에서 초성, 중성, 종성의 구조를 유지하지 않는 한글형태를 말하고, 옛한글은 구결을 포함하지 않은 옛한글 음절을 표현 정도를 의미하고, 형태소 분석에서 효율성은 곧 바로 형태소 분석을 할 수 있는 정도를 말한다. 한자 표현은 유니코드 규정에서는 대략20000만자가 넘지만 현재로서는 실제로 지원하는 운영체제는 없지만 차츰 지원할 것으로 간주한다.

.................................................................................................... [*◀ 차 례*](http://www.copyeditor.co.kr/reference/char-font/%EC%A1%B0%ED%95%A9%ED%98%95_%EC%99%84%EC%84%B1%ED%98%95_%EC%9C%A0%EB%8B%88%EC%BD%94%EB%93%9C.htm#▶ 차례 ◀)

**▶ 참고 문헌**

The Unicode Standard, Version 2.0, The Unicode Consortium, 1996

국어 어휘 데이터베이스 구축에 대한 연구, 정광, 국립국어연구원, 1996

국제 표준 문자 코드 제안 한자에 대한 연구, 서경호, 문화체육부, 1997.11.30.

남북한 한글코드 비교조사 및 통일방안 기초연구, 서정수, 문화체육부, 1995.12.

프로그래머 바이블, 오상문, 영진출판사, 1994.1.20.

컴퓨터 속의 한글, 이준희·정내권, (주)정보시대, 1991.

컴퓨터와 한글의 만남, 임현모, 정보문화사, 1992.

한글 정보산업의 현황과 발전 전망 - 제 7회 한글 및 한국어 정보처리 학술대회, 강태진, 1995.

한글공학, 오길록 외, 대영사, 1994.9.10.

한글코드에 관한 연구, 홍윤표, 국립국어연구원, 1995.

한글코드와 자판에 관한 기초 연구 최종 보고서, ㈜한글과 컴퓨터, 문화부, 1992.12.

확장 완성형 코드 시스템, 마이크로소프트

"KSC 5700"의 제정 의미와 전망, http://ktmp.kaist.ac.kr/~geenie/news/hangul.html, 1998.09.08.

ISO/IEC-10646 Universal Multiple Octet Coded Character Set(UCS)에 대하여, 정주원, world wide web, 1995.11.11.

KSC5700코드의 문제점, 제 1차 한글 코드 표준화 위원회 회의록world wide web, 1997.6.24.

MS사의 "확장완성형" 한글 코드에 대한 한컴의 입장(2) , world wide web, 1995.5.

MS사의 [한글 윈도우 95]에서 채택키로 한 통합형 한글 코드에 대한 ㈜한글과컴퓨터사의 입장, 한글과컴퓨터사, 1995.5.

Windows95나 NT에서의 한글 FAQ, world wide web

WWW의 국제화와 한글, http://simac.kaist.ac.kr/~jwjung/seminar/www-18n/i18n.html

공진청, KS한글코드체계 제정, http://ktmp.kaist.ac.kr/~geenie/news/hangul.html, 1998.09.08.

소프트웨어의 국제화, 방준영, http://intranet.www-kr.org/workshop/ws4/D15/

제4차 인터넷 한글표준화 협의회 회의록, http://standard.nca.or.kr/hangul/meeting4.html

한글 Windows 98에서 지원되는 Code는, http://www.microsoft.com/korea/windows98/basics/faq/default.htm, 1998.08.

한글 코드에 대하여, 정주원, world wide web, 1995.11.12.

한자·한글 변환기 원형 개발에 관한 연구, 유재수, world wide web

한컴, 유니코드기술위원회 참가, 11172자 한글 코드 영역 확보, world wide web, 1995.5.

.............................................................................................