# 유니 코드를 바탕으로 한 프로그램 상에서의 한글 자/모 구별방법에 따른 연구

권 훈°, 김정희, 곽호영
제주대학교 컴퓨터 공학과
{dreamerz°, carina, kwak}@cheju.ac.kr

# Reserches to divided Hangul Spelling for Program based on Uni-Code

Hoon-Kwon°, Jeong-Hee Kim, Ho-Young Kwak
Dept. of Computer Engineering, Cheju National University

**요 약**

본 논문에서는 컴퓨터상에서의 처리되는 한글 코드 중 유니 코드를 이용한 한글 입력방법을 분석하여, 이를 일련의 프로그램에서 초·중·종성에 따라 자/모음을 분리, 구별하는 방법을 제안하고, 이에 따라, 제안된 방법을 적용한 구현 프로그램을 통해 정확한 자/모음 분리를 할 수 있었으며, 한글 입력에서의 정확도와 신뢰도 향상 및 단어에 따른 정확한 조사판별이 가능해 짐을 알 수 있었다. 또한, 제안방법에 따른 통계적 데이터를 가지고 각종 분야에 적용이 가능하게 됨을 알 수 있었다.

## 1. 서 론

현재 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. [4,5] 컴퓨터의 보급이 점차 증가되기 시작하면서 한국어로 된 명령과 한국어를 처리할 수 있는 프로그램이 필요하게 되었다. 초창기 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 각각의 한글 처리 방법을 고안하게 되었다. [1,2,3]

본 논문에서는, 현실적으로 사용하고 있는 한글 코드에 대해 분석하여 보고, 유니 코드를 이용한 일련의 프로그램에서(여기서는 VisualBasic) 한글 자/모음의 구별/분리방법을 제안하고, 제안한 방법에 의해 간단한 프로그램 구현하여 이를 통한 효율적 데이터 처리 및 활용방법에 대해 논의해 보고자 한다.

본 논문의 구성은 먼저, 2장에서는 지금까지 제정되거나 사용되었던 한글 코드 방식에 대해 살펴보고, 3장에서는 유니 코드를 기반으로 한글 코드 자/모음의 분리 방법을 제시하고 기술한다. 4장에서는 3장의 방법을 통해, 실제 프로그램에 적용하여 봄으로써 이에 따른 기대 효과를 분석하며, 마지막으로 5장에서 연구에 따른 결과 및 향후 연구를 기술한다.

## 2. 한글 코드의 종류와 원리 [6]

1) 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. 이는 7비트 아스키 환경에서도 사용할 수 있다는 장점이 있지만 한글 처리에서는 적합하지 않다.

2) 3바이트 한글 코드

3바이트 한글 코드는 한글 한 음절을 초성, 중성, 종성으로 나누고 각각 한 바이트씩 배정하여 처리하는 것으로 한글 한 음절을 처리하기 위해서 3바이트를 사용한다. 일종의 조합형 코드이지만 한글 1음절을 표현할 때 2바이트 조합형 코드보다 1바이트를 더 많이 사용하기 때문에 비효율적이다.

3) KSC5601 완성형 한글

한글을 음절 단위로 순서대로 배치하여 각각의 코드값을 부여하는 방식이다. 구현 방식을 보면 첫 번째 바이트와 두 번째 바이트 모두 164 ~ 256까지의 영역만을 사용한다. 따라서, 한글표현의 한계 및 한글의 자소 분리, 입출력문제 등을 가진다.

4) 상용 조합형 한글코드

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. 완성형 코드의 문제를 다소 개선하였다.

5) 유니 코드

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

1995년에 이르러 유니 코드2.0을 제정하게 된다. 유니 코드2.0에서는 한글은 두 가지 영역으로 배정받았다. 첫 번째는 현대 한글 11172자를 완성형 방식의 코드순으로 배열한 것인데, 현대 한글 11172자는 완성형

처럼 완성된 음절을 기준으로 배정하였다. KSC5601완성형 코드와는 다르게 일정한 조합 규칙을 유지하고 있어서 사용하기 편리하다. 두 번째는 초성, 중성, 종성을 자소 단위로 배정한 조합형 방식 코드로 2바이트 상용 조합형이나 KSC5601-1992 표준 조합형처럼 고정된 길이가 아니라 자소 개수만큼 사용하는 N바이트 형식으로 지정하였다. 국내에서는 유니 코드 2.00이 발표되자마자 KSC5700이라는 이름으로 유니 코드 2.0을 국가 표준 코드로 채택하게 된다.

### 3. 유니 코드 기반의 자/모음 분리 방법

#### 3.1 유니 코드 구성

유니 코드의 구성표는 [표. 1]과 같다.

실제로 '가'를 기반으로 한 유니 코드 구성표는 [표. 2]에 표현되어 있다.

#### 3.2 유니 코드 기반의 자/모의 분리방법 제안

본 논문에서는 유니 코드 기반은 자/모 분리를 구현하기 위하여 VisualBasic 6.0을 이용하였다.

[표. 1]의 초·중·종성에 해당하는 코드값을 해당 한글에 대한 유니 코드 값에서 찾아주면 되는데, 이를 위한 방법은 다음과 같이 구현가능하다. 예를 들어 [표. 2]에서의 '갰'이라는 글자를 초·중·종성으로 나눠보면 [표. 3] 과 같이 분석할 수 있다.

이를 기반으로 실제 프로그램에서의 분리 방법을 제안한 것이 [표. 4] 이다.

표 3. '갰' 코드의 분석

```
'갰'의 분석   "가"의 기본코드는 "AC00"이다.
    초성값 : 0
    중성값 : 1
    종성 값 : 12
```

표 4. 초·중·종성 구별방법 제안

```
base=AscW("가")
code = AscW("갰") - base        ' code=40
초성 = code ₩ (21 * 28)         ' 초성=0
code = code Mod (21 * 28)       ' code=40
중성 = code ₩ 28                ' 중성=1
종성 = code Mod 28              ' 종성=12
```

이와 같은 방법으로, 유니 코드에서의 각 한글코드에 대한 초·중·종성에 따른 자/모 구별이 가능해진다.

표 2. '가' 기반의 유니코드 구성표

| | | | | | |
|---|---|---|---|---|---|
| 가 | AC00(AC00+0) | | 갊 | AC0A( | 10) |
| 각 | AC01( | 1) | 갋 | AC0B( | 11) |
| 갂 | AC02( | 2) | 값 | AC0C( | 12) |
| 갃 | AC03( | 3) | 갍 | AC0D( | 13) |
| 간 | AC04( | 4) | 갎 | AC0E( | 14) |
| 갅 | AC05( | 5) | 갏 | AC0F( | 15) |
| 갆 | AC06( | 6) | 감 | AC10( | 16) |
| 갇 | AC07( | 7) | 갑 | AC11( | 17) |
| 갈 | AC08( | 8) | 값 | AC12( | 18) |
| 갉 | AC09( | 9) | 갓 | AC13( | 19) |
| 갔 | AC14( | 20) | 갞 | AC1E( | 30) |
| 강 | AC15( | 21) | 갟 | AC1F( | 31) |
| 갖 | AC16( | 22) | 갠 | AC20( | 32) |
| 갗 | AC17( | 23) | 갡 | AC21( | 33) |
| 갘 | AC18( | 24) | 갢 | AC22( | 34) |
| 같 | AC19( | 25) | 갣 | AC23( | 35) |
| 갚 | AC1A( | 26) | 갤 | AC24( | 36) |
| 갛 | AC1B( | 27) | 갥 | AC25( | 37) |
| 개 | AC1C( | 28) | 갦 | AC26( | 38) |
| 객 | AC1D( | 29) | 갧 | AC27( | 39) |
| | | | **갨** | **AC28(** | **40)** |

### 4. 제안한 방법에 따른 실제 프로그램 구현

#### 4.1 구현 배경

실제 프로그램 상에서 간단한 단문 연습 프로그램을 구현하던 중, 완성형 한글 코드로 구현을 하다 보니 [표. 5]와 같은 문제점이 발생하여 정확도 및 신뢰도에 대한 검증이 필요할 것으로 생각되어 제안 방법을 통한 구현을 하게 되었다.

#### 4.2 제안 방법에 따른 구현

제안 방법에 따라 간단한 단문 연습 프로그램을 구현해 보았다. [그림. 1]에서 보듯이 각 한글코드를 각각 초·중·종성에 따른 자/모 구별을 하여 타자 연습 시에 참고할 수 있도록 통계적 데이터를 제공해 주고, 이에 따른 속도와 정확도를 표시하였다. 또한 각 글쇠에 누

표 1. 유니 코드 표

| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 초성 | ㄱ | ㄲ | ㄴ | ㄷ | ㄸ | ㄹ | ㅁ | ㅂ | ㅃ | ㅅ | ㅆ | ㅇ | ㅈ | ㅉ | ㅊ | ㅋ | ㅌ | ㅍ | ㅎ | | | | | | | | | |
| 중성 | ㅏ | ㅐ | ㅑ | ㅒ | ㅓ | ㅔ | ㅕ | ㅖ | ㅗ | ㅘ | ㅙ | ㅚ | ㅛ | ㅜ | ㅝ | ㅞ | ㅟ | ㅠ | ㅡ | ㅢ | ㅣ | | | | | | | |
| 종성 | | ㄱ | ㄲ | ㄳ | ㄴ | ㄵ | ㄶ | ㄷ | ㄹ | ㄺ | ㄻ | ㄼ | ㄽ | ㄾ | ㄿ | ㅀ | ㅁ | ㅂ | ㅄ | ㅅ | ㅆ | ㅇ | ㅈ | ㅊ | ㅋ | ㅌ | ㅍ | ㅎ |

른 횟수를 [그림. 2]에서 보여주고 있다.

표 5. 한글 음소에 따른 한글처리시 문제점

| |
|---|
| 1. 건 빵 -> ㄱ ㅓ ㄴ sp ㅃ ㅏ ㅇ -- A<br>건 빵 -> ㄱ ㅏ bs ㅓ ㄴ sp ㅃ ㅏ ㅇ -- B |
| 2. 사용자 이름 입력시 조사표현의 단일화 문제<br>권 훈 (은)는 -- A 김정희 (은)는 -- B |

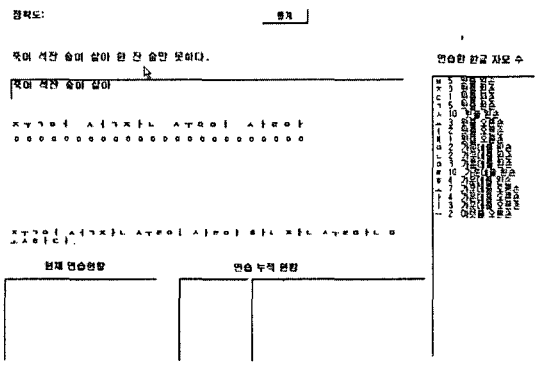

그림 1. 제시 방법에 따른 타자연습 구현

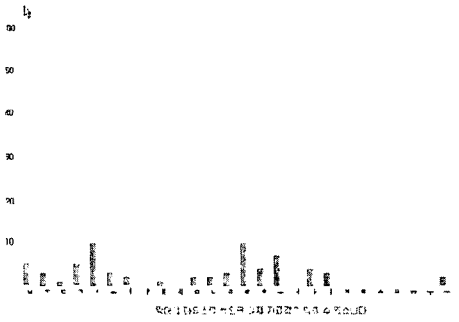

그림 2. 각 자모에 따른 통계

## 5. 결 론

[표. 5]에서 나타난 바와 같은 문제점 중 첫 번째에 대한 비교를 위해 아래와 같은 가정해 보았다.

| |
|---|
| 정 타 : 발 없는 말이 천리 간다.<br>오타 수정 : "말" 입력시 "머 -> 말"로 수정<br>"간" 입력시 "거 -> 간"로 수정 |

위 가정을 통해 직접 입력한 결과는 [표. 6]에서 보는 바와 같이 음소단일 처리와 자/모 분리간의 정확도 연에서 차이를 나타내었다.

따라서, 제안방법에 따른 결론은 [표. 7]에서 요약된 것과 같이 한글 처리의 정확도 측정에 있어서, 신뢰도가 증가되며, 한글 입력시의 단어에 대한 조사판별 문제 역시 제시 방법을 이용하면 쉽게 가능함을 알 수 있었다.

본 논문에서는 유니코드를 기반으로 한 각 한글코드에 대한 초·중·종성에 따른 자/모 구별을 이용하여 간단한 실제 단문연습에서 적용하여 각 글쇠를 빈도 및 자/모음 입력에 따른 신뢰도 및 정확도를 향상 시키고, 또한 각 글쇠에 대한 데이터를 통계자료화 하여 글쇠 입력자에 대한 보다 나은 글쇠 입력 방법과 정확도를 제시하였다.

표 6. [표. 5]의 첫 번째 비교 결과

| 비 교 | | 정 타 | 오타 수정 |
|---|---|---|---|
| 음소단일처리 | 정확도 | 100% | 100% |
| | 속도 | 414타/분 | 254타/분 |
| 자/모분리처리 | 정확도 | ***100%*** | ***87.9%*** |
| | 속도 | 414타/분 | 254타/분 |

표 7. 제안방법에 따른 결론

| 방법 / 문제점 | 음소단일 처리 | 자/모 분리처리 |
|---|---|---|
| 정확도/신뢰도 | 화면상의 결과 판별<br>정확도/신뢰도 감소 | 각 자/모별 판별<br>정확도/신뢰도 증가 |
| 조사판별문제 | 단일 표기 | 종성의 유무판별<br>정확한 조사 적용 |

이와 같은 방법에 의해 산출된 통계적 데이터를 기반으로 하여 사용자의 한글 입력 패턴 및 오타 발생빈도 및 자주 사용하는 글쇠 등을 파악하여 또 다른 응용의 데이터로 활용이 가능하다.

또한, 어떤 단어에 대한 조사판별에 있어서도 단어 맨 마지막 글자의 종성 유무판별에 의한 정확한 조사 적용이 가능함을 알 수 있었다.

따라서, 제시 방법을 이용하면, 한글 타자 연습이나 워드프로세서뿐만 아니라, 한글 처리에 따른 자/모 구별이 필요한 분야에 통계적 자료로 보다 많은 애플리케이션에서 활용될 수 있을 것이다.

## 참 고 문 헌

[1] 컴퓨터와 한글의 만남, 임현모, 정보문화사, 1992.
[2] 한글코드에 관한 연구, 홍윤표, 국립국어연구원, 1995.
[3] 확장 완성형 코드 시스템, 마이크로소프트
[4] 공진청, KS한글코드체계 제정, http://ktmp.kaist.ac.kr/~geenie/news/hangul.html, 1998.
[5] 한글코드와 자판에 관한 기초 연구 최종 보고서, ㈜한글과 컴퓨터, 문화부, 1992.12.
[6] 한글 및 한국어 정보 처리 코드, 전상훈 (한국어정보처리연구소), 99. 1.