L2/09-013

INTERNATIONAL STANDARD

ISO/IEC 14651

Second edition
2007-12-01

AMENDMENT 1
2008-10-24

Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering

AMENDMENT 1

Technologies de l'information — Classement international et comparaison de chaînes de caractères — Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable d'ordre de classement

AMENDEMENT 1

Reference number
ISO/IEC 14651:2001/Amd.1:2003(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 14651:2007 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 2, *Coded character sets*.

Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering

AMENDMENT 1

Page 1, Clause 1

Replace the second bullet and its notes by:

- A Common Template Table. A given tailoring of the Common Template Table is used by the reference comparison method. The Common Template Table describes an order for all characters encoded in ISO/IEC 10646:2003 up to Amendment 4. It allows for a specification of a fully deterministic ordering. This table enables the specification of a string ordering adapted to local ordering rules, without requiring an implementer to have knowledge of all the different scripts already encoded in the UCS.

  NOTE 1 This Common Template Table is to be modified to suit the needs of a local environment. The main worldwide benefit is that, for other scripts, often no modification is required and the order will remain as consistent as possible and predictable from an international point of view.

  NOTE 2 The character repertoire used in this International Standard is equivalent to that of the Unicode Standard version 5.1.

Page 2, Clause 3

Replace the normative references with the following.

ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 10646:2003/Amd.1:2005, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Amendment 1: Glagolitic, Coptic, Georgian and other characters*

ISO/IEC 10646:2003/Amd.2:2006, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Amendment 2: N'Ko, Phags-pa, Phoenician and other characters*

ISO/IEC 10646:2003/Amd.3:2008, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Amendment 3: Lepcha, Ol Chiki, Saurashtra, Vai and other characters*

ISO/IEC 10646:2003/FDAmd.4:2008, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Amendment 4: Cham, Game Tiles, and other characters*

Page 6, Clause 6.2.2

Add the following note after the first paragraph:

NOTE Collating elements with more characters have preference over shorter ones. As an example, if a multi-character collating element is defined for “abc” and another one is defined for “ab” or another one for ‘bc”, then if “abc” is encountered, the collating element for “abc” will apply and not the one for “ab” or “bc”.

Page 12, Clause 6.3.2

Replace WF 4 with the following:

WF 4. A *tailored_table* shall contain exactly one *order_start* statement *followed later by exactly one order_end statement*. The *order_start* statement shall appear after the *symbol_definition* entries and before the *symbol_weight* entries after all reordering of lines has been applied.

Delete line beginning with WF 10. (This will imply renumbering WF 11, WF 12 and WF 13 for the next edition).

Replace the beginning of WF 12 with the following:

WF 12. Any *symbol_range* shall contain two *symbol*s which meet the following conditions: 1) Each of the two *symbol*s shall contain a common prefix of length one. The prefix can be any letter except 'U'.

Page 13, clause 6.3.3, 5th line from the bottom of the page

Replace “beenreordered” with “been reordered”.

Page 16, Subclause 6.5

Replace Subclause 6.5 with the following.

6.5 Name of the Common Template Table and name declaration

The name ISO14651_2008_TABLE1 shall be used whenever the Common Template Table is referred to externally as a base point in a given context, whether in a process, contract, or procurement requirement-. If another name is used due to practical constraints, a declaration of conformance shall indicate the correspondence between this other name and the name ISO14651_2008_TABLE1.

The use of a defined name is necessary to manage the different stages of development of this table. This follows from the nature of the reference character repertoire, for which development will be ongoing for a number of years or even decades.

Page 17, Annex A

Replace Annex A with the following.

Annex A
(normative)

Common Template Table

In order to minimize formatting problems and the risk of errors in reproduction, the common template table is provided separately in a machine-readable file as a normative component of this International Standard. The file name for this language version is different from the normative reference name specified in 6.5 of this International Standard due to the existence of file versions commented in other natural languages. The file for this language version can also be retrieved on the ITTF web site at the following URL:

http://www.iso.org/ittf/ISO14651_2008_TABLE1_En.txt

There is an official French version of the file which only differs in its comments (its technical content is identical), and its name is: ISO14651_2008_TABLE1_fr.txt

NOTE 1 This International standard deprecates, but does not preclude specific reference to, the previous tables, which contained and still contain ordering information applicable to the repertoires of previous versions of ISO/IEC 10646 and their amendments. The previous tables can be found at the following URLs:

[ordering information on the repertoire of characters as defined in ISO/IEC 10646-1:1993 including Amendments 1-9] http://www.iso.org/ittf/ISO14651_2000_TABLE1.htm

[ordering information on the combined repertoire of characters of ISO/IEC 10646-1:2000 and ISO/IEC 10646-2:2001] http://www.iso.org/ittf/ISO14651_2002_TABLE1_en.txt

[ordering information on the repertoire of characters as defined in ISO/IEC 10646:2003] http://www.iso.org/ittf/ISO14651_2003_TABLE1_en.txt

[ordering information on the repertoire of characters as defined in ISO/IEC 10646-1:2003 including Amendments 1-2] http://www.iso.org/ittf/ISO14651_2006_TABLE1_en.txt

The current Common Template Table reflects the repertoire of characters as defined in ISO/IEC 10646:2003 up to its amendment 4, as indicated in the scope.

NOTE 2 The repertoire targeted by this International standard is equivalent to the repertoire of *The Unicode Standard Version 5.1, published by The Unicode Consortium*.

When ordering data applicable to other amendments of ISO/IEC 10646:2003 becomes available, this International Standard and specifically its Common Template Table will be amended accordingly to cover the ordering of the additional characters and scripts. To meet cultural requirements of specific communities, delta declarations will have to be applied to the amended table as defined in this International Standard.

ISO_14651_2008_TABLE1 is the name that is used for referring to this table in this version of this International Standard.

Page 20, replace in the unordered list the word "notres" by the word "notre".

Page 28, add the following clause C.4:

C.4 A proposed method of preprocessing Hangul (or Hangeul)

C.4.1 Introduction

The CTT does not formally include weights for Hangul syllables. As a result, some tailoring of the table and/or preprocessing of strings is required in order to collate Hangul data according to the algorithm specified in this standard.

One method is simply to give modern Hangul syllable characters primary weights in increasing sequence and to normalize any input strings containing conjoining jamos so that all input strings contain only preformed Hangul syllable characters. This method is quick and efficient for data containing only modern Hangul, because the Hangul syllables are already encoded in the correct collation order for Korean.

For data containing Old Hangul, the situation is more complicated, because no Unicode normalization form provides input strings that can simply be weighted element□by□element to produce appropriate keys for collation. Further preprocessing may be necessary to produce keys that can be used for the desired collation behavior.

The essential issue is that the desired Hangul collation order is a syllabic order, but Hangul syllables are built up from a sequence of three jamos: a syllable□initial, a syllable□peak (= vowel), and an optional syllable□final. Each of those jamos, in turn, may consist of one to three subparts, particularly in Old Hangul, which has numerous consonant or vowel clusters represented by single jamo characters.

The basic strategy for handling such Hangul data is to first ensure that it is represented entirely in jamos, so that there is no mixture in the input of conjoining jamos with preformed Hangul syllable characters. This step can be accomplished using Unicode Normalization Form NFD to decompose any preformed Hangul syllable characters. Then a Korean syllable boundary determination algorithm is used to identify all syllabic boundaries in the data.

Once all the Korean syllabic boundaries are determined in the input data, the initial, peak, and final jamos for each syllable can be weighted so as to provide keys which, when compared, give the desired results for syllabic ordering of the strings.

Ideally, Old Hangul data preprocessed to decompose it and make syllabic□boundary determinations will contain exactly one initial jamo, one peak jamo, and optionally, one final jamo for each syllable. However, certain kinds of input data might result, when decomposed, in sequences containing more than one initial conjoining jamo, and so on. In such cases, Hangul preprocessing may involve an additional mapping step that ensures that any such sequence of jamos is first mapped to the corresponding single initial jamo intended to represent that consonant cluster (of two or three subparts) for Old Hangul. The same considerations apply for any sequences of peak jamos or final jamos in the data.

There are various strategies for weighting the initial, peak, and final jamos of the preprocessed Korean text to produce the desired syllabic order. One approach is to expand each initial, peak, and final jamo into a sequence of three weights, based on the internal composition of each jamo. Simple jamos get one weight; two□part jamos get two weights; three□part jamos get three weights. Any remaining positions in the nine weights for each syllable are filled with EMPTY (U0000) weights.

For example, for the preformed Hangul syllable U+AC01, the data would first be preprocessed into the jamo sequence <1100, 1161, 11A8> and then be weighted as the following key:

U1100 U0000 U0000 U1161 U0000 U0000 U11A8 U0000 U0000

For a sequence containing a multi□part Old Hangul jamo representing a cluster, the keys would have multiple values. For example, for the input sequence <1123, 1161, 11A8>, the U+1123 would be given weights by its three subparts as follows:

U1107 U1109 U1103 U1161 U0000 U0000 U11A8 U0000 U0000

The same kind of weight expansion would be done for any multi□part peak or final jamo as well.

When a Korean syllable contains no final jamo, the last three weights are all set to EMPTY

(U0000).

With each Korean syllable expanded to nine weights by this preprocessing and weighting scheme, weights for each syllable are lined up correctly on syllabic boundaries, and direct comparison of the resulting keys produces the correct collation results.

Although this weighting strategy works for all Hangul data, including modern Hangul and Old Hangul, it produces much□expanded keys which are not efficient for production applications of collation. Once syllabic boundaries are determined by preprocessing of Korean data, alternative approaches to weighting the jamos can produce much more compact keys which also produce the same end results for collation.

It is also possible for a collation implementation to do the equivalent of this syllabic preprocessing for Hangul data on the fly while weighting an input string, so it is not technically required to have a formally separate preprocessing step for Hangul which converts all of the input data into a preprocessing form first before weighting it for comparison. This is particularly important for incremental comparison algorithms, which are very performance□sensitive, and which typically cannot afford to preprocess entire strings before starting to do incremental comparison of them.

C.4.1.1 BNF

What follows specifies one particular set of rules for transforming Hangul data in UCS so that Hangul can be properly collated by ISO/IEC 14651-supporting program.

Since we will specify the transforming rules in a widely used notation, called a context-free grammar (or grammars, for short) or BNF (for Backus-Naur Form or Backus-Normal Form), we will briefly introduce BNF.

The following explanations come from [Compilers, Principles, Techniques, and Tools. Aho, Sethi, and Ullman. Addison-Wesley Publishing Company. 1985]. Some parts are slightly edited so that we can better understand in ISO/IEC 14651 context.

For example, an if-else statement in C has the form

```
if (expression) statement else statement
```

The if-else statement is the concatenation of the keyword if, an opening parenthesis, an expression, a closing parenthesis, a statement, the keyword else, and another statement. The structure can be expressed in BNF as

```
<stmt> -> if ( <expr> ) <stmt> else <stmt>
```

in which the arrow may be read as "can have the form". Such a rule is called a production. The keyword if and the parentheses are called "tokens". <expr> and <stmt> represent a sequence of tokens and are called non-terminals.

A context-free grammar has four components:

1) A set of "tokens", known as "terminal symbols".

2) A set of "non-terminals".

3) A set of productions where each production consists of a non-terminal, called the left side of the production, and arrow ("->"), and a sequence of tokens and/or non-terminals, called the right side of the production.

4) A designation of one of the non-terminals as the start symbol.

One specifies the transformation rules (or grammars) by listing their productions, with the productions for the start symbol listed first.

Here, non-terminals are shown enclosed within a pair of brackets, e. g., <si>, <si1>, <si2>, <si3>. Terminals are shown without brackets, e. g., U1100, U1162, where U1100 is HANGUL CHOSEONG KIYEOK and U1162 is HANGUL JUNGSEONG AE.

Productions with the same non-terminal on the left can have their right sides grouped, with the alternative right sides separated by the vertical bar symbol "|", which we read as "or".

Example 1.1. Consider expressions consisting of two digits separated by plus or minus signs, e. g., 9 + 2, and 3 - 1. The following grammar describes the syntax of these expressions. The productions are:

```
 <expr> -> <term> + <term>                          (production 1a)

  <expr> -> <term> - <term>                          (production 1b)

  <term> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9    (production 1c)
```

The right sides of the two productions with non-terminal <expr> on the left side can equivalently be grouped:

<expr> -> <term> + <term> | <term> - <term>

<expr> and <term> are non-terminals with <expr> being the starting non-terminal because its productions are given first. +, -, 0, 1, ..., and 9 are terminals (or tokens).

A grammar derives strings by beginning with the start symbol and repeatedly replacing a non-terminal by the right side of a production for that non-terminal. The strings that can be derived from the start symbol form the language defined by the grammar.

Example 1.2. The language defined by the grammar of Example 1.1 consists of two digits separated by a plus or minus sign.

The ten productions for the non-terminal <digit> allow it to stand for any of the 0, 1, ..., 9. From production 1c, a single digit by itself is a term. Productions 1a and 1b express the fact that if we take any digit and follow it by a plus or minus sign and then another digit we have an expression.

a) 9 is a <term> by production 1c

b) 9 - 5 is an <expr> by production 1b, since 9 is a <term> and 5 is also a <term>

Example 1.3. The Latin alphabet as used in English consists of 26 letters; in English 5 letters are vowels and the others are consonants. That can be expressed as follows:

<latin-alphabet-en> -> <vowel> | <consonant>

<vowel> -> a | e | i | o | u

<consonant> -> b | c | d | f | g | h | j | k | l | m | n | p | q | r | s | t | v | w | x | y | z

C.4.1.2 Syntax-directed translation

A translation scheme is a context-free grammar in which program fragments called "semantic actions" are embedded within the right sides of productions. The position at which an action is to be executed is shown by enclosing it between braces ("{ }") and writing it within the right side of a production, as in

```
<expr> -> <term> + <term> { print('+') }      (production e1)

<expr> -> <term> - <term> { print('-') }      (production e2)

<term> -> 0 {print ('0')}                     (production t0)

<term> -> 1 {print ('1')}                     (production t1)

...

<term> -> 9 {print ('9')}                     (production t9)
```

A translation scheme generates an output (using the 'print' command) for each sentence x generated by the underlying grammar by executing the actions in the order they appear.

The above translation scheme translates a given expression into postfix form. This scheme accepts expressions having only two numbers and a plus or minus in between. For example, '9 + 5' or '9 - 5' is accepted, but '1 + 2 - 3' or '9 - 8 - 7' is not.

Expressions such as 3 + 5 or 9 - 8 are called infix notation, since a plus or minus sign, which is a binary operator, are written between two numbers. With a postfix notation, the binary operator (a plus or minus sign) is put after two numbers.

For example, the postfix notation for 3 + 5 is 3 5 + (plus sign is put 'after' two numbers, 'not between' two numbers).

Let's see how 9 - 5 is translated into 9 5 -. We start with the production e1 "<expr> -> <term> + <term> {print('+')}". The first part of the right side is <term>. Then the production t9 "<term> -> 9" matches ’9‘ and '9' is printed. Now '+' of the right side does not match with '-'. Therefore we give up e1 "<expr> -> <term> + <term> {print('+')}".

Now we try the next production e2 "<expr> -> <term> - <term> {print('-')}". The production t9 "<term> -> 9" matches with '9' and prints '9'. Then '-' in the production matches with '-‘; however nothing is printed at this point. Now the production t5 "<term> -> 5" matches with '5' and prints '5'.

The production e2 "<expr> -> <term> - <term> {print('-')}" matches with the given string '9 - 5'. At this point, '-' is printed. We are done. Therefore, the final output is '9 5 -', which is a postfix notation for the given expression '9 - 5'.

C.4.2 Examples showing how to transform data

In Section 1, we studied the basic concept of BNF and translation scheme. With this background, let's see examples showing how to transform data in Hangul.

C.4.2.1 Example 1 (a simple example using only a few Hangul characters)

- For simplicity, we included only two syllable-initial characters, two syllable-peak characters, two syllable-final characters, and two fill characters.

- Some exercises are explained below to show how input characters are transformed according to the given rules.

- Two notations are used below.

Character strings within '/*' and '*/' are comments and therefore do not affect the transforming process in any way. They are just for humans.

'==' defines constants. For example, if you have 'SI-FILL == U115F', you can substitute U115F for SI-FILL whenever you see SI-FILL. By using SI-FILL instead of U115F (a code position which is hard to memorize), we can improve readability.

Example 1 (abridged from Example 2; this Example is for demo purpose)

/* constants */

SI-FILL == U115F /* syllable-initial FILL character */

```
SP-FILL == U1160 /* syllable-peak    FILL character */


/* Hangul syllables */

/* LS: Left side */

/* RS: Right side or pattern */

/* we start from <root> */

/* FAIL cancels temporary OUTPUT */

/* 'finalize OUTPUT' finalizes temporary OUTPUT */

/* action is shown within { }.  */

/* Most actions are to output some characters. */


/* rule R01B accepts four combinations of characters:

  U1100 U1161 | U1100 U1163 |  U1102 U1161 | U1102 U1163

            KA            KYA            NA          NYA */


---- -------    -----------------------------------

rule   LS        RS (or pattern)

---- -------    -----------------------------------

ROOT <root>   -> <hg-syl> {finalize OUTPUT}

R01B <hg-syl> -> <si> <sp> |

R01F            SI-FILL { print('SI-FILL') } <sf>


/* <si> : syllable-initial letters

  <si1>: syllable-initial simple letters */

R11D <si>  -> <si1>

R12A <si1> -> U1100 { print('U1100') } |

R12B         U1102 { print('U1101') }

          /* rules R12A and R12B: output without any transform */


/* <sp> : Syllable-peak letters
```

```
   <sp1>: syllable-peak simple letters */

R21D <sp>  -> <sp1>

R22A <sp1> -> U1161 { print('U1161') } |

R22B        U1163 { print('U1163') }

            /* rules R22A and R22B: output without any transform */


/* <sf> : syllable-final letters

   <sf1>: syllable-final simple letters */

R31D <sf>  -> <sf1>

R32A <sf1> -> U11A8 { print('U11A8') } |

R32B        U11AB { print('U11AB') }

            /* rules R32A and R32B: output without any transform */
```

Exercise 1.1 Suppose that the input string is "U1100 U1161" (Hangul syllable "GA").

- An input string represents Hangul syllable "GA".
- Unless "FAIL" is mentioned below, the pattern match succeeds.
- When "finalize OUTPUT" is executed, a temporary output becomes final.

- Using the above rules, we will process the input string.
- We start with rule ROOT "<root> -> <hg-syl>".
- Then we go to the first rule with its left side <hg-syl>, i.e., rule R01B "<hg-syl> -> <si> <sp>". Now we try the first component of this rule, which is "<si>".
- Then we go to rule R11D "<si> -> <si1>", which is the only rule with its left side <si>.
- Then we go to the first rule with its left side <si1>, i.e., rule R12A "<si1> -> U1100 { print ('U1100') }". Its right side "U1100" matches input character U1100. At this point, 'U1100' is printed by the action { print ('U1100') } in rule R12A.

Now we are done with rule R12A "<si1> -> U1100".

Then we back up to R11D "<si> -> <si1>". We are done with rule R11D.

Then we back up to R01B "<hg-syl> -> <si> <sp>". We are done with <si>, the first component of the right side in R01B.

Now we are ready to try the second component of the right side in R01B, i.e., <sp>.

We go to rule R21D "<sp> -> <sp1>", which is the only rule with its left side <sp>.

Then we go to the first rule with its left side <sp1>, i.e., rule R22A "<sp1> -> U1161 { print ('U1161') }". Its right side "U1161" matches input character U1161. At this point, 'U1161' is printed by the action { print('U1161') } in rule R22A.

Now we are done with rule R22A "<sp1> -> U1161 { print ('U1161') }".

Then we back up to rule R21D "<sp> -> <sp1>". We are done with rule R21D.

Then we back up to rule R01B "<hg-syl> -> <si> <sp>". Since, we are done with the second component of the right side, i.e., <sp>, at this point, we are done with rule R01B "<hg-syl> -> <si> <sp>".

Then we back up to rule ROOT "<root> -> <hg-syl> {finalize OUTPUT}". We are done with rule ROOT <root>. At this point, temporary OUTPUT is finalized.

- In this exercise, we do not change anything. We just try to match the rules against the input string and then print out without any transformation. The above process can be summarized as follows:

```
----------   ------------            -----------------

rule (LHS)   MATCH/FAIL                    OUTPUT by actions

----------   ------------            -----------------

ROOT <root>

R01B <hg-syl>

R11D <si>

R12A <si1>   MATCH R12A <si1> -> U1100          U1100

             MATCH R11D <si> -> <si1>

             MATCH R01B <hg-syl> -> <si>

R21D <sp>

R22A <sp1>   MATCH R22A <sp1> -> U1161          U1161

             MATCH R21D <sp> -> <sp1>

             MATCH R01B <hg-syl> -> <si> <sp>

             MATCH ROOT <root> -> <hg-syl>      finalize output
```

* final output: U1100 U1161

Exercise 1.2 Suppose that the input string is "U115F U11A8" (*HANGUL JONGSEONG KIYEOK*).

```
----------   ------------              -----------------
rule (LHS)   MATCH/FAIL                     OUTPUT by actions
----------   ------------              -----------------
ROOT <root>
R01B <hg-syl>
R11D <si>
R12A <si1>    FAIL  R12A <si1> -> U1100
              FAIL  R12B <si1> -> U1102
              FAIL  R11D <si> -> <si1>
              FAIL  R01B <hg-syl> -> <si>

R01F <hg-syl>  MATCH SI-FILL {print ('U115F')}      U115F
R31D <sf>
R32A <sf1>    MATCH R32A <sf1> -> U11A8          U11A8
              MATCH R31D <sf> -> <sf1>
              MATCH R01F <hg-syl> -> SI-FILL <sf>.
              MATCH ROOT <root> -> <hg-syl>       finalize output
```

* final output: U115F U11A8

C.4.2.2 Example 2

- This example transforms one Hangul syllable into 9 code positions: 3 code positions for each of syllable-initial, syllable-peak, and syllable-final character, respectively.

- Some EMPTY (U0000) characters are intentionally inserted so that we can collate Hangul properly (this is especially useful for collating Old Hangul properly).

Example 2

```
/* constants */

SI-FILL == U115F /* syllable-intial fill character */

SP-FILL == U1160 /* syllable-peak   fill character */


/* Hangul syllables */

---- -------    -----------------------------------

rule   LS         RS (or pattern)

---- -------    -----------------------------------

ROOT <root>   -> <hg-syl> { finalize OUTPUT }

R01B <hg-syl> -> <si> <sp> { print('U0000 U0000 U0000') }

             /* FAIL cancels relevant temporary output */

R01F            SI-FILL { print('SI-FILL U0000 U0000 U0000 U0000 U0000') }

              <sf>


/* syllable-initial letters:  <si1> a syllable-initial simple letter */

R11D <si>  -> <si1> { print('U0000 U0000') }

R12A <si1> -> U1100 { print('U1100') } |

R12B          U1102 { print('U1102') }  /* output without any transform */


/* syllable-peak letters:  <sp1> a syllable-peak simple letter; */

R21D <sp>  -> <sp1> { print('U0000 U0000') }
```

```
R22A <sp1> -> U1161 { print('U1161') } |
R22B        U1163 { print('U1163') } /* output without any transform */


/* syllable-final letters:  <sf1> a syllable-final simple letter; */
R31D <sf>  -> <sf1> { print('U0000 U0000') }
R32A <sf1> -> U11A8 { print('U11A8') } |
R32B        U11AB { print('U11AB') } /* output without any transform */
```

Exercise 2.1 Suppose that the input string is "U1100 U1161" (Hangul syllable "GA").

```
----------   ------------              -----------------
rule (LHS)   MATCH/FAIL                      OUTPUT by actions
----------   ------------              -----------------


ROOT <root>
R01B <hg-syl>
R11D <si>
R12A <si1>    MATCH R12A <si1> -> U1100          U1100
              MATCH R11D <si> -> <si1>          U0000 U0000
              MATCH R01B <si>
R21D <sp>
R22A <sp1>    MATCH R22A <sp1> -> U1161          U1161
              MATCH R21D <sp> -> <sp1>          U0000 U0000
              MATCH R01B <hg-syl> -> <si> <sp>    U0000 U0000 U0000
              MATCH ROOT <root> -> <hg-syl>      finalize output


* final output: U1100 U0000 U0000   U1161 U0000 U0000   U0000 U0000 U0000
```

Exercise 2.2 Suppose that the input string is "U115F U11A8".

- input file represents HANGUL JONGSEONG KIYEOK.

```
----------    ------------                  -----------------
rule (LHS)    MATCH/FAIL                    OUTPUT by actions
----------    ------------                  -----------------
ROOT <root>
R01B <hg-syl>
R11D <si>
R12A <si1>    FAIL  R12A <si1> -> U1100
              FAIL  R12B <si1> -> U1102
              FAIL  R11D <si> -> <si1>
              FAIL  R01B <hg-syl> -> <si>
R01F <hg-syl>  MATCH <hg-syl> -> SI-FILL          U115F U0000 U0000
                                             U0000 U0000 U0000
R31D <sf>
R32A <sf1>    MATCH R32A <sf1> -> U11A8          U11A8
              MATCH R31D <sf> -> <sf1>          U0000 U0000
              MATCH R01F <hg-syl> -> SI-FILL <sf>
              MATCH ROOT <root> -> <hg-syl>       finalize output

* final output: U115F U0000 U0000   U0000 U0000 U0000   U11A8 U0000 U0000
-----------------------------------------------------------------------
```

C.4.3 Preprocessing Modern and Old Hangul

C.4.3.1 Preprocessing Modern Hangul syllables (U+AC00 ~ U+D7A3)

Each of the code positions in the range of U+AC00 ~ U+D7A3 corresponds to one Modern Hangul syllable. Before applying the rules in Section C.4.3.2, we need to decompose each code position in the range of U+AC00 ~ U+D7A3 into two or three code positions.

Code positions corresponding to syllables with a syllable-final letter will be decomposed into three code positions. Code positions corresponding to syllable-initial, syllable-peak and syllable-final letters are concatenated in that order.

Code positions corresponding to syllables without a syllable-final letter will be decomposed into two code positions. Code positions corresponding to syllable-initial and syllable-peak letters are concatenated in that order.

Code positions for syllable-initial, syllable-peak and optional syllable-final letters are computed as follows:

```
 a code position for a syllable-initial letter = 0x1100 + (c / 588)

 a code position for a syllable-peak    letter = 0x1161 + (c % 588) / 28

 a code position for a syllable-final   letter =
   if (c % 28 != 0) then (0x11A8 - 1) + c % 28
   else none
```

Note. '/' is an integer division operator and '%' is a modulo operator.

C.4.3.2 Preprocessing Jamo (U1100 ~ U11FF)

The following rules can preprocess 11,172 Modern Hangul syllables and other incomplete syllables represented using Jamo (U1100 ~ U11FF)

```
/* constant */
SI-FILL == U115F
SP-FILL == U1160


/* Hangul syllable */
/* complete syllables: R01A and R01B */
/* incomplete syllables: R01C, R01D, R01E and R01F */


ROOT <root>   -> <hg-syl> {finalize OUTPUT}
R01A <hg-syl> -> <si> <sp> <sf> |
R01B            <si> <sp> { print('U0000 U0000 U0000') } |
R01C            <si> SP-FILL
              {print('SP-FILL U0000 U0000 U0000 U0000 U0000') } |
R01D            SI-FILL { print('SI-FILL U0000 U0000') } <sp> <sf> |
R01E            SI-FILL { print('SI-FILL U0000 U0000') } <sp>
              { print('U0000 U0000 U0000') } |
R01F            SI-FILL { print('SI-FILL U0000 U0000 U0000 U0000 U0000') }
             <sf>



/* syllable-initial letters:
  <si1> a syllable-initial simple letter
  <si2> a syllable-initial 2-complex letter (composed of 2 simple letters)
  <si3> a syllable-initial 3-complex letter (composed of 2 simple letters)
*/


R11A <si>  -> <si1> <si1> <si1> |
R11B         <si1> <si1> { print('U0000') } |
R11C         <si1> <si2> |
R11D         <si1> { print('U0000 U0000') } |
R11E        <si2> <si1> |

R11F        <si2> { print('U0000') }

/* R11G <si3>: no si3 for modern Hangul */


R12A <si1> -> U1100 { print('U1100') } |

R12B        U1102 { print('U1102') } |

R12C        U1103 { print('U1103') } |

R12D        U1105 { print('U1105') } |

R12E        U1106 { print('U1106') } |

R12F        U1107 { print('U1107') } |

R12G        U1109 { print('U1109') } |

R12H        U110B { print('U110B') } |

R12I        U110C { print('U110C') } |

R12J        U110E { print('U110E') } |

R12K        U110F { print('U110F') } |

R12L        U1110 { print('U1110') } |

R12M        U1111 { print('U1111') } |

R12N        U1112 { print('U1112') }


/* output without any transform */


R13A <si2> -> U1101 { print('U1100 U1100') } |

R13B        U1104 { print('U1103 U1103') } |

R13C        U1108 { print('U1107 U1107') } |

R13D        U110A { print('U1109 U1109') } |

R13E        U110D { print('U110C U110C') }


/* R14 <si3> ->  no si3 for modern Hangul */


/* Syllable-peak letters:
  <sp1> a syllable-peak simple letter

  <sp2> a syllable-peak 2-complex letter (composed of 2 simple letters)

  <sp3> a syllable-peak 3-complex letter (composed of 3 simple letters) */


R21A <sp>  -> <sp1> <sp1> <sp1> |

R21B         <sp1> <sp1> { print('U0000') } |

R21C         <sp1> <sp2> |

R21D         <sp1> { print('U0000 U0000') } |

R21E         <sp2> <sp1> |

R21F         <sp2> { print('U0000') } |

R21G         <sp3>


R22A <sp1> -> U1161 { print('U1161') } |

R22B         U1163 { print('U1163') } |

R22C         U1165 { print('U1165') } |

R22D         U1167 { print('U1167') } |

R22E         U1169 { print('U1169') } |

R22F         U116D { print('U116D') } |

R22G         U116E { print('U116E') } |

R22H         U1172 { print('U1172') } |

R22I        U1173 { print('U1173') } |

R22J         U1175 { print('U1175') }


R23A <sp2> -> U1162 { print('U1161 U1175') } |

R23B         U1164 { print('U1163 U1175') } |

R23C         U1166 { print('U1165 U1175') } |

R23D         U1168 { print('U1167 U1175') } |

R23E         U116A { print('U1169 U1161') } |

R23F         U116C { print('U1169 U1175') } |

R23G         U116F { print('U116E U1165') } |
R23H         U1171 { print('U116E U1175') } |
R23I        U1174 { print('U1173 U1175') }

R24A <sp3> -> U116B { print('U1169 U1161 U1175') }
R24B         U1170 { print('U116E U1165 U1175') }



/* syllable-final letters:
  <sf1> a syllable-final simple letter
  <sf2> a syllable-final 2-complex letter (composed of 2 simple letters)
  <sf3> a syllable-final 3-complex letter (composed of 3 simple letters) */

R31A <sf>  -> <sf1> <sf1> <sf1> |
R31B         <sf1> <sf1> { print('U0000') } |
R31C         <sf1> <sf2> |
R31D         <sf1> { print('U0000 U0000') } |
R31E         <sf2> <sf1> |
R31F         <sf2> { print('U0000') }
/* R31G <sf3>: no sf3 for modern Hangul */



R32A <sf1> -> U11A8 { print('U11A8') } |
R32B         U11AB { print('U11AB') } |
R32C         U11AE { print('U11AE') } |
R32D         U11AF { print('U11AF') } |
R32E         U11B7 { print('U11B7') } |
R32F         U11B8 { print('U11B8') } |
R32G         U11BA { print('U11BA') } |
R32H         U11BC { print('U11BC') } |
R32I        U11BD { print('U11BD') } |
R32J        U11BE { print('U11BE') } |

R32K        U11BF { print('U11BF') } |

R32L        U11C0 { print('U11C0') } |

R32M        U11C1 { print('U11C1') } |

R32N        U11C2 { print('U11C2') }  /* output without any transform */


R33A <sf2> -> U11A9 { print('U11A8 U11A8') } |

R33B        U11AA { print('U11A8 U11BA') } |

R33C        U11AC { print('U11AB U11BD') } |

R33D        U11AD { print('U11AB U11C2') } |

R33E        U11B0 { print('U11AF U11A8') } |

R33F        U11B1 { print('U11AF U11B7') } |

R33G        U11B2 { print('U11AF U11B8') } |

R33H        U11B3 { print('U11AF U11BA') } |

R33I        U11B4 { print('U11AF U11C0') } |

R33J        U11B5 { print('U11AF U11C1') } |

R33K        U11B6 { print('U11AF U11C2') } |

R33L        U11B9 { print('U11B8 U11BA') } |

R33M        U11BB { print('U11BA U11BA') }


/* R34  <sf3> ->    no sf3 for modern Hangul */
```

Preprocessing Old Hangul is very similar to preprocessing Modern Hangul.

C.4.4 Conclusions

The current normative clauses of ISO/IEC 14651 cannot directly collate Hangul data in ISO/IEC 10646, especially Old Hangul data. Without preprocessing incorrect results are achieved.

What precedes proposes a method of preprocessing Hangul data in ISO/IEC 10646 so that the output can be used as an input to ISO/IEC 14651-conformant program, which will then collate Hangul properly.

Rules in Section C.4.3.2 transform one modern Hangul syllable (including incomplete syllables) into 9 code positions. When Hangul data is transformed this way, it can be collated properly by ISO/IEC 14651-conformant program.

Rules in Section C.4.3.2 can be easily extended to preprocess Old Hangul data accoding to its collating rules.

ICS 35.060

Price based on 3 pages