1. thaiconv.c

```
/*
  This is the file thaiconv.c of the CJK macro package ver. 4.3.0
  (20-Jun-1999).

  Thai preprocessor

  based on the program cttex ver 1.15 written by
  Vuthichai Ampornaramveth <vuthi@ctrl.titech.ac.jp>

  It uses a dictionary to find break points between words.

  This version is for special use with cjk-enc.el (and MULEenc.sty) of
  the CJK package for LaTeX 2e:

    each line will be started and ended with special commands which
    start and end a Thai environment.

    each processed character (and breakpoints between Thai words) will be
    treated similarly.
*/

/* Maximum length of input line */
```

#define MAXLINELENGTH 1000

/* Maximum number of WORDS in one line */
#define MW 60

/* Maximum number of words to LOOKBACK */
#define BACKDEPTH 3

/* Characters to be skipped */
#define SKIPWORD(x) $(((x) < 128) \lor (((x) \le {}^{\#}\mathtt{F9}) \land ((x) \ge {}^{\#}\mathtt{F0})))$

/* HIGH Chars */
#define HIGHWORD(x) $(((x) \ge 128))$

/* Check level of a character */
#define NOTMIDDLE(x) $((x) < {}^{\#}\mathtt{D0}\ ?\ 0 : (\mathit{levtable}[(x) - {}^{\#}\mathtt{D0}] \ne 0))$

/* the internal word break character */
#define CUTCODE 254

2.

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* the dictionary */
#include "thaiconv.h"

```
void dooneline(unsigned char *, unsigned char *);
void savestatus(int *, int *, int *, int *, int *, int *,
unsigned char *, int);
int mystrncmp(unsigned char *, unsigned char *, int);
int findword(unsigned char *, int *);
int countmatch(unsigned char *in, unsigned char *out);
void adj(unsigned char *);
void filter(unsigned char *);
int moveleft(int);
```

3.

```
/* Table Look-Up for level of a character */

int levtable[] =
 {
0, 2, 0, 0, 2, 2, 2, 2, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 2, 3, 3, 3, 3, 3, 2, 3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0
 };
int lefttab[] =
 {
136, 131,        /* Meaning : change 136 to 131, ... */
137, 132,        /* Up Level Mai Ek, To, Ti ... */
138, 133,
139, 134,
140, 135,
#ED, #8F,        /* Circle */
#E8, #98,        /* Top Level Mai Ek, To, Ti, ... */
#E9, #99,
#EA, #9A,
#EB, #9B,
#EC, #9C,
#D4, #94,        /* Sara I, EE, ... */
#D5, #95,
#D6, #96,
#D7, #97,
#D1, #92,
#E7, #93
 };
/* the character code of a visible representation of a wordbreak */
  int cutcode;
/* to avoid zillions of signedness warnings */
  unsigned char **wordptr = (unsigned char **) Wordptr;
```

4.

```
int main(int argc, char *argv[])
{
  FILE *fp, *fopen();
  unsigned char str[MAXLINELENGTH], out[MAXLINELENGTH];
  int i, c;

  cutcode = CUTCODE;

  if (argc > 1)
    sscanf(argv[1], "%d", &cutcode);

  fp = stdin;

  while (!feof(fp))
   {
    (void) fgets((char *) str, MAXLINELENGTH - 1, fp);

    if (!feof(fp))
     {
      dooneline(str, out);

      if (argc > 1)
        printf("%s", out);                              /* for testing purposes */
      else
       {
        adj(out);                                       /* Choose appropriate WANNAYUK */

      /* Mule command #57 (begin Thai) */
        printf("\17757\177\177");

        i = 0;
        while ((c = out[i++]))
          if (c == '\n')
          /* Mule command #58 (end Thai) */
            printf("\17758\177\177\n");
          else if (!HIGHWORD(c))
            putchar(c);
          else if (c == CUTCODE)
          /* Mule command #61 (Thai word break) */
            printf("\17761\177\177");
          else
          if (NOTMIDDLE(c))
             /* Mule command #60 (print Thai vowel/tone) */
            printf("\17760\177%d\177", c);
          else
             /* Mule command #62 (print Thai consonant with glue) */
            printf("\17762\177%d\177", c);
       }
     }
   }

  return 0;
}
```

5.

```
void dooneline(unsigned char *in, unsigned char *out)
{
  int i, j, k, l, old_i;
  int wlist[MW], poslist[MW], jlist[MW], windex, pos, fence, backmode;
  int prev_error = 0;

  i = old_i = j = l = 0;
  windex = 0;
  fence = 0;
  backmode = 0;
  prev_error = 0;
  while (in[i])
  {
  /* old_i is the value of i before looking back mode */
  /* i > old_i means looking back was successful, cancel it */
    if (i > old_i)
      backmode = 0;
    if (SKIPWORD(in[i]))
    {                                              /* Chars to be skipped ? */
      if (prev_error)
      {
      /* Mark words not in dict */
        out[j++] = cutcode + 1;
        prev_error = 0;
      }
      backmode = fence = windex = 0;                 /* Begin new word list */
      while (SKIPWORD(in[i]) ∧ in[i])                /* Skip non-Thai chars */
        out[j++] = in[i++];                          /* and Thai numbers */
    }
    if (in[i])                                       /* Still not EOL ? */
      do
      {
        if ((k = findword(in + i, &pos)))
        {                                            /* Found a word in dict */
          if (prev_error)
          {
            out[j++] = cutcode + 1;
            prev_error = 0;
          }
          wlist[windex] = i;
          poslist[windex] = pos;
          jlist[windex] = j;
          windex++;
        /* For speed, limit the number of words to LOOK back */
        /* by creating a fence */
        /* Fence may only INCREASE */
          if (windex − BACKDEPTH > fence)
            fence = windex − BACKDEPTH;
```

```
    for (l = 0; l < k; l++)                                    /* Copy word */
      out[j++] = in[i++];
  /* Mai Ya Mok & Pai Yan Noi */
  /* I was using this code before adding these two characters
     into the dict. Now they are in the dict and I no longer need
     these two lines.
  */
    while ((in[i] ≡ #E6) ∨ (in[i] ≡ #CF))
      out[j++] = in[i++];
    if (!SKIPWORD(in[i]))               /* Make sure it's not the last Thai word */
      out[j++] = cutcode;                            /* Insert word sep symbol */
   }
  else
   {                                                            /* Not in Dict */
  /* Shortening the prev words may help */
  /* Try to Look Back */
    while ((windex > fence) ∧ !k)
     {
    /* Save status before looking back */
      if (!backmode)
       {
        backmode = 1;
        savestatus(&windex, wlist, poslist, jlist, &i, &j, out, 1);
        old_i = i;
       }
      pos = poslist[windex − 1] − 1;                        /* Skip back one word */
      while ((pos ≥ 0) ∧
      (l = countmatch(wordptr[pos] + 1, in + wlist[windex − 1])))
       {
        if ((l ≡ wordptr[pos][0]) ∧
        !NOTMIDDLE(in[wlist[windex − 1] + l]))
         {
          k = 1;
          break;
         }
        pos −−;
       }
    /* A shorter version of prev word found */
      if (k)
       {
        out[j = jlist[windex − 1] + l] = cutcode;
        poslist[windex − 1] = pos;
        j++;
        i = wlist[windex − 1] + l;
       }
      else
       {
        if (backmode ∧ (windex ≡ fence + 1))
         {
        /* Search-Back method can't help, restore prev status */
          savestatus(&windex, wlist, poslist, jlist, &i, &j, out, 0);
```

```
                break;
              }
            windex--;
          }
        }
      /* Sure that word is not in dictionary */
      if (k ≡ 0)
      {
        prev_error = 1;                    /* Begin unknown word area */
        out[j++] = in[i++];                /* Copy it */
        backmode = fence = windex = 0;     /* Clear Word List */
      }
    }
  } while ((k ≡ 0) ∧ (!SKIPWORD(in[i])));
}
out[j] = 0;
/* Filter out words not in dict */
filter(out);
}
```

6.

```
#if 0
/* Sequential version */
int findword(unsigned char *in)
{
  int i;
  for (i = NUMWORDS − 1; i ≥ 0; i−−)
  {
    if (mystrncmp(in, wordptr[i] + 1, wordptr[i][0]) ≡ 0)
    {
      printf("Found:␣%s␣%d\n", wordptr[i] + 1, wordptr[i][0]);
      return wordptr[i][0];
    }
  }
  return 0;
}
#endif
```

7.

```
/* Calling: Index to a string
   Return : Length of recognized word, and position of that word in
            dictionary

   Binary search method.
*/

  int findword(unsigned char *in, int *pos)
  {
    int up, low, mid, a, l;
    up = NUMWORDS − 1;                                      /* Upper bound */
    low = 0;                                                /* Lower bound */
    *pos = −1;                                           /* If word not found */
  /* Found word at the boundaries ? */
    if (mystrncmp(in, wordptr[up] + 1, wordptr[up][0]) ≡ 0)
    {                                              /* last (longest) match found */
      *pos = up;
      return wordptr[up][0];
    }
    if (mystrncmp(in, wordptr[low] + 1, wordptr[low][0]) ≡ 0)
      mid = low;                                  /* first (shortest) match found */
    else
    {                                                   /* Begin binary search */
      do
      {
        mid = (up + low)/2;
        a = mystrncmp(in, wordptr[mid] + 1, wordptr[mid][0]);
#if 0
        printf("%d␣%d␣%d␣%s\n", low, mid, up, wordptr[mid] + 1);
#endif
        if (a ≠ 0)
        {
          if (a > 0)
            low = mid;
          else
            up = mid;
        }
      } while ((a ≠ 0) ∧ (up − low > 1));
      if (a ≠ 0)
      {                                                       /* Word not found */
        mid−−;                                       /* Go back in dictionary */
      /* Can we find a shorter word ? */
        if (!countmatch(wordptr[mid] + 1, in))
          return 0;                                      /* No character matches */
```

```
      while (mid ∧ (l = countmatch(wordptr[mid] + 1, in)))
      {
        if ((l ≡ wordptr[mid][0]) ∧ !NOTMIDDLE(in[l]))
        {
          *pos = mid;
          return l;
        }
        mid--;
      }
      if (a)
        return 0;                                   /* really no match */
    }
  }
  up = mid;
  if (up < NUMWORDS)
    do
    {                                       /* Find the longest match */
      up++;
      a = mystrncmp(in, wordptr[up] + 1, wordptr[up][0]);
      if (a ≡ 0)
        mid = up;
    } while ((a ≥ 0) ∧ (up < NUMWORDS − 1));
#if 0
  printf("Found␣:␣%s␣%d\n", wordptr[mid] + 1, wordptr[mid][0]);
#endif
  *pos = mid;
  return wordptr[mid][0];
}
```

8.

```
int countmatch(unsigned char *in, unsigned char *out)
{
  int i;
  i = 0;
  while (in[i] ≡ out[i])
    i++;
  return i;
}
```

9.

```
void savestatus(int *windex, int *wlist, int *poslist, int *jlist,
int *oi, int *j, unsigned char *out, int mode)
{
  static int lwindex, lwlist[MW], lposlist[MW], ljlist[MW], li, lj;
  int i;
  static unsigned char lout[MAXLINELENGTH];
#if 0
```

```
    printf ("Save␣call␣%d\n", mode);
#endif
    if (mode)
    {                                                    /* Save */
      lwindex = *windex;
      for (i = 0; i < lwindex; i++)
      {
        lwlist[i] = wlist[i];
        lposlist[i] = poslist[i];
        ljlist[i] = jlist[i];
      }
      for (i = 0; i < *j; i++)
        lout[i] = out[i];
      li = *oi;
      lj = *j;
    }
    else
    {                                                    /* Restore */
      *windex = lwindex;
      for (i = 0; i < lwindex; i++)
      {
        wlist[i] = lwlist[i];
        poslist[i] = lposlist[i];
        jlist[i] = ljlist[i];
      }
      for (i = 0; i < lj; i++)
        out[i] = lout[i];
      *oi = li;
      *j = lj;
    }
  }
```

10.

```
/*
  Thai version of strncmp: b must be the word from dictionary.
*/

  int mystrncmp(unsigned char *a, unsigned char *b, int l)
  {
    int i;
    i = strncmp((char *) a, (char *) b, l);
    if (i)
      return i;
    else
      return NOTMIDDLE(a[l]);
  }
```

11.

```
/* What to do with words outside dictionary */

void filter(unsigned char *line)
{
  int i, j, c, a, found;
  unsigned char str[MAXLINELENGTH];

  strcpy((char *) str, (char *) line);
  found = i = 0;
  a = -1;
  while ((c = str[i]))
  {
    if (c == cutcode)
      a = i;
    else if (c == cutcode + 1)
    {
      found = 1;
      if (!SKIPWORD(str[i + 1]))
        str[i] = cutcode;
      if (a >= 0)
      {
        str[a] = cutcode + 1;
        a = -1;
      }
    }
    else if (SKIPWORD(c))
      a = -1;

    i++;
  }
  if (found)
  {
    i = j = 0;
    while ((c = str[i++]))
      if (c != cutcode + 1)
        line[j++] = c;

    line[j] = 0;
  }
}
```

12.

```
void adj(unsigned char *line)
{
  unsigned char top[MAXLINELENGTH];
  unsigned char up[MAXLINELENGTH];
  unsigned char middle[MAXLINELENGTH];
  unsigned char low[MAXLINELENGTH];

  int i, k, c;
```

```
/* Split string into 4 levels */
/* Clear Buffer */
  for (i = 0; i < MAXLINELENGTH; i++)
    top[i] = up[i] = middle[i] = low[i] = 0;
  i = 0;
  k = -1;
  while ((c = line[i++]))
   {
    if (k >= 0)
     {
      switch ((c > #D0) ? levtable[c - #D0] : 0)
       {
        case 0:                                          /* Middle */
      /* Special Case for Sara-Am */
          if (c == #D3)
           {
            up[k] = #ED;
            middle[++k] = #D2;                           /* Put Sara-Ar */
           }
          else
            middle[++k] = c;
          break;
        case 1:                                          /* Low */
          low[k] = c;
          break;
        case 2:                                          /* Up */
          up[k] = c;
          break;
        case 3:                                          /* Top */
          top[k] = c;
          break;
       }
     }
    else
      middle[++k] = c;
   }
/* Beauty Part Begins */
  for (i = 0; i <= k; i++)
   {
  /* Move down from Top -> Up */
    if ((top[i]) ∧ (up[i] == 0))
     {
      up[i] = top[i] - 96;
      top[i] = 0;
     }
```

```
/* Avoid characters with long tail */
  if (middle[i] ≡ #BB ∨                                  /* Por Pla */
  middle[i] ≡ #BD ∨                                      /* For Far */
  middle[i] ≡ #BF)                                       /* For Fun */
   {
    if (up[i])
      up[i] = moveleft(up[i]);
    if (top[i])
      top[i] = moveleft(top[i]);
   }
/* Remove lower part of TorSanTan and YorPhuYing if necessary */
  if (middle[i] ≡ #B0 ∧ low[i])                          /* TorSanTan */
    middle[i] = #9F;
  if (middle[i] ≡ #AD ∧ low[i])                          /* YorPhuYing */
    middle[i] = #90;
/* Move lower sara down , for DorChaDa, TorPaTak */
  if (middle[i] ≡ #AE ∨
  middle[i] ≡ #AF)
   {
    if (low[i])
      low[i] = low[i] + 36;
   }
 }
/* Pack Back To A Line */
 i = 0;
 k = 0;
 while (middle[i])
  {
   line[k++] = middle[i];
   if (low[i])
     line[k++] = low[i];
   if (up[i])
     line[k++] = up[i];
   if (top[i])
     line[k++] = top[i];
   i++;
  }
/* Number of bytes might change */
 line[k] = 0;
}
```

13.

```
int moveleft(int c)
{
 int i;
```

```
  for (i = 0; i < 34; i += 2)
   {
    if (lefttab[i] ≡ c)
       return lefttab[i + 1];
   }
  return c;
}
```

14.

```
/* end of thaiconv.c */
```

15. Index.

a: 7, 10, 11.
adj(): 2, 4, 12.
argc: 4.
argv: 4.
b: 10.
BACKDEPTH: 1, 5.
backmode: 5.
c: 4, 11, 12, 13.
countmatch(): 2, 5, 7, 8.
cutcode: 3, 4, 5, 11.
CUTCODE: 1, 4.
dooneline(): 2, 4, 5.
fence: 5.
feof(): 4.
fgets(): 4.
filter(): 2, 5, 11.
findword(): 2, 5, 6, 7.
fopen(): 4.
found: 11.
fp: 4.
HIGHWORD(): 1, 4.
i: 4, 5, 6, 8, 9, 10, 11, 12, 13.
in: 2, 5, 6, 7, 8.
j: 5, 9, 11.
jlist: 5, 9.
k: 5, 12.
l: 5, 7, 10.
lefttab: 3, 13.
levtable: 1, 3, 12.
li: 9.
line: 11, 12.
lj: 9.
ljlist: 9.
lout: 9.
low: 7, 12.
lposlist: 9.
lwindex: 9.
lwlist: 9.
main(): 4.
MAXLINELENGTH: 1, 4, 9, 11, 12.
mid: 7.
middle: 12.
mode: 9.
moveleft(): 2, 12, 13.
MW: 1, 5, 9.
mystrncmp(): 2, 6, 7, 10.
NOTMIDDLE(): 1, 4, 5, 7, 10.
NUMWORDS: 6, 7.
oi: 9.
old_i: 5.
out: 2, 4, 5, 8, 9.
pos: 5, 7.
poslist: 5, 9.
prev_error: 5.
printf(): 4, 6, 7, 9.
putchar(): 4.
savestatus(): 2, 5, 9.
SKIPWORD(): 1, 5, 11.
sscanf(): 4.
stdin: 4.
str: 4, 11.
strcpy(): 11.
strncmp(): 10.
top: 12.
up: 7, 12.
windex: 5, 9.
wlist: 5, 9.
wordptr: 3, 5, 6, 7.
Wordptr: 3.

The thaiconv program

	Section	Page
thaiconv.c	1	1
Index	15	14

This document was produced with the `c2cwebx` program