TR-IT-0243

# Chinese Speech Synthesis within CHATR

Ming Yue Xie-Zhang

November, 1997

ABSTRACT

This report describes my work of six months on Chinese speech synthesis within CHATR at ATR Interpreting Telecommunications Research Laboratories (ITL). It contains the following chapters: Chapter 2 gives a brief introduction to Chinese language. Chapter 3 describes the Chinese speech database and the work involved in creating a CHATR Chinese database. Chapter 4 presents the problems encountered in Chinese synthesis and the solutions to them. In this chapter, I also explain the changes which had been made to the CHATR source code in order to achieve a better result on the segmental level. Chapter 5 describes my work on automatic conversion from Chinese orthography to Chinese phonetic symbols (Pin-yin) using a lexicon. Chapter 6 discusses some remaining problems and the future work for Chinese synthesis.

# Acknowledgments

I'd like to thank all members of ITL for their kindness and their help during my stay. I'd like to thank especially

- Norio Higuchi and Nick Campbell for the opportunity they gave me to work at ITL,
- Nick Campbell for technical guides and stimulating discussions,
- the members of department 2 for their helpful work about CHATR and friendly discussion during lunch time,
- Wen Ding, Qiang Huo and Li Deng for their helpful discussions about Chinese synthesis.

# Contents

# Chapter 1

# Introduction

This report describes my work of six months on Chinese speech synthesis within CHATR at ATR Interpreting Telecommunications Research Laboratories (ITL). CHATR is a generic speech synthesis system, developed by Dept 2 of ITL. For a general introduction to CHATR as well as an overview of the system architecture, please refer to the manual [Weeks 97].

When I arrived at ITL, the system already worked for Japanese, English, German and Korean languages. Chinese is a new language for CHATR system. So, I had to start my work from the first step of speech synthesis, namely creating a new language database (Chinese) for CHATR. Then, I worked on how to improve Chinese synthesis.

In this report, at first I will briefly introduce Chinese language: the written Chinese and the spoken Chinese. Then, I will describe the Chinese database and the work involved in creating a CHATR Chinese database. Then, I will present the problems encountered in Chinese synthesis and propose some solutions to them. I will also explain the changes which had been made to the CHATR source code in order to achieve a better result on the segmental level. Then, I will describe my work on automatic conversion from Chinese orthography to Chinese phonetic symbols (Pin-yin) using a lexicon. Finally, I will discuss some remaining problems and the future work for Chinese synthesis.

# Chapter 2

# Chinese Language

## 2.1 Written Chinese

Chinese is not a phonetic but ideographical language. The Chinese orthography is called **Hanzi**. There are about 7000 characters. The written Chinese exists in two forms: simplified Chinese (used in Mainland) and the traditional Chinese (used in Taiwan and HongKong). In the computer, each character is coded with two bytes, it exists three types of coding:

- **GB code:** simplified Chinese used in Mainland
- **Big5 code:** traditional Chinese used in Hongkong and Taiwan
- **EUC code:** traditional Chinese used in Taiwan

## 2.2 Spoken Chinese

With the same written Chinese, there are thousands of dialects in China. One may be very different from the others. I will present here only the Chinese official speaking: Mandarin or Pu-tong-hua, defined by Pin-yin system (Chinese phonetic system).

The Pin-yin system consists of 21 initial parts and 40 final parts:

**Initial parts:** b p m f d t n l g k h
j q x zh ch sh r z c s

**Final parts:** a o e ai ei ao ou an en ang eng ong
i ia ie iao iou-iu ian iang in ing iong
↪ *yi ya ye yao you yan yang yin ying yong*
u ua uo uai uei-ui uan uen-un uang ueng
↪ *wu wa wo wai wei wan wen wang weng*
ü üe üan ün
↪ *yu yue yuan yun*
er (ch)i (c)i ng ê

The pronunciation of each character is one syllable, corresponds to a combination of an initial part and a final part or to a final part alone. When a final part is not preceded by an initial part, in writing the letter "i" becomes "y", the letter "u" becomes "w", and the letter "ü" becomes "yu".

In the spoken Chinese, there are in total 410 different syllables. And each one has usually 5 different tones (see the figure 2.1). So, a character's pronunciation corresponds to a toned syllable.

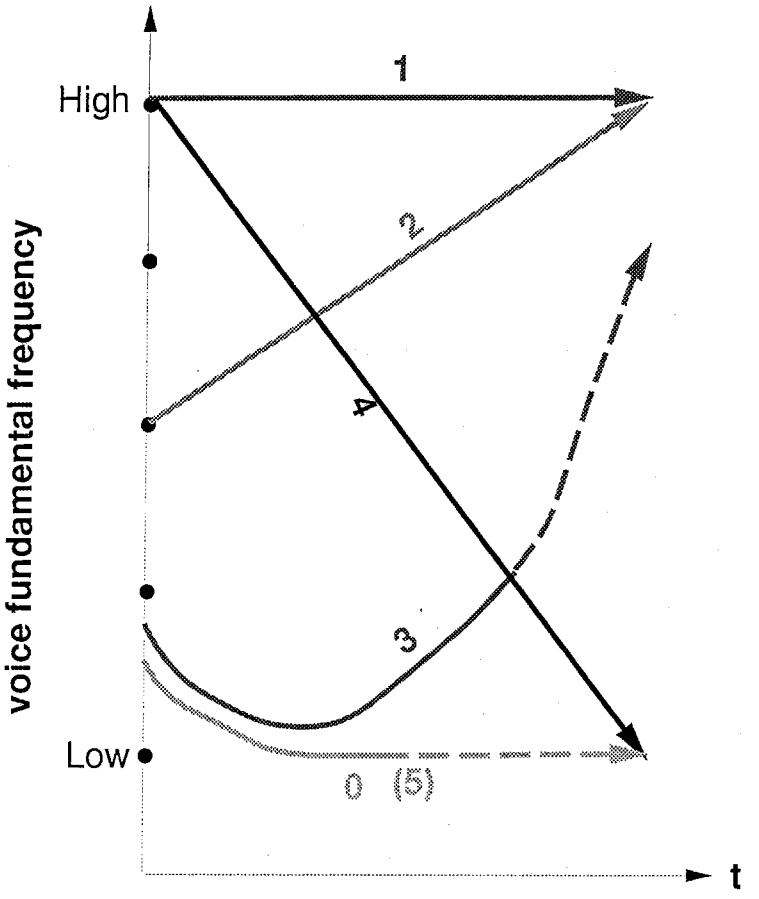


Figure 2.1: Chinese tones illustration

In Chinese language, there are some ambiguities: the different characters may have the same pronunciation, and the same character may have the different pronunciations (also different meanings). So sometimes, we need to know the context for determining the meaning of a character.

# Chapter 3

# Chinese Database

## 3.1 The HKU CD-ROM Databases

The databases used for Chinese synthesis is issued from a Mandarin Corpus CD-ROM Version. This database is constructed at the Speech Laboratory of the Department of Computer Science, the University of HongKong in 1993-96. A total of 20 native Mandarin speakers have been employed to read prompting messages displayed on a monitor screen. The data are stored in five CD-ROMs. Please refer to [HKU96] for the further information.

The HKU databases contain the following data types:

- **Isolated Syllables:** all Mandarin syllables in all tones
- **Words:** 11 words of 2 to 4 syllables are selected in such a way that their pronunciations include all the Mandarin phones.
- **Digit Strings:** 16 digit strings of 4 to 7 digits each are designed to exhaust all inter-digit triphones.
- **Rhymed Syllables:** 3 sentences of 7 syllables each are so designed that all syllables in the first sentence rhyme with /a/, those in second rhyme with /i/ and those in third rhyme with /u/.
- **Continuous Speech:** hundreds of lines text with unique contents
- **Retroflexed Ending Words:** A set of words with and without retroflexed endings have been read by two speakers.

Corresponding to each utterance, be it an isolated syllable or continuous speech, the information stored includes the orthographic transcription (the Chinese characters in Big5 code) with the toned Pin-yin symbols, the phonetic labels of the waveform, and the digitized waveform itself. These three kinds of information are stored in files of three categories: the 'txt' files are text files for orthographic transcriptions, the 'lab' files are text files for phonetic labels, and the 'wfm' files are binary files for digitized waveforms. Each utterance is stored in separate 'wfm' file. All the orthographic transcriptions and the phonetic labels of the utterances spoken by each speaker are grouped into two files, 'txt' and 'lab'.

Because of the time limitation, I have selected two speakers speech data (a male (4m) and a female (9f)). I have used only the continuous speech for the Chinese synthesis within CHATR. These two databases are named "`cs9f`" and "`cs4m`". For the database cs4m, there are 975 utterances, about 45 minutes speech; and for the database cs9f, there are 973 utterances, about one hour speech.

Please refer to the Appendix B about the procedure to collect a Chinese database from the CD-ROMs.

## 3.2 Problems found in the HKU databases

In the HKU databases, the phonetic labels are based on the initial parts and some semi-triphones. The alignment of the phonetic labels has been realized by auto-alignment.

The following shows the phoneme set used in the databases.

- the initial parts
- the semi-triphones:

```
a       a<i>
o       o<u>
e       <i>e    e<i>     e<n>     e<ng>
<ch>i   <c>i     <b>i
u       yv       er       ng
```

I have used this original database to make the first Chinese synthesis database to see how well it works. I have found the synthesis results are not intelligible due to the following reasons:

- Chinese is a toned language, but in the label files, the tone information is missing
- many phonetic labels are not well segmented (maybe about 50%)
- the same phoneme is used for the initial part "n" and the "n" in the final parts (but they are different in Chinese language)

## 3.3 Build a new database

We have already seen the problems of the original database, we need to build a new Chinese database.

### 3.3.1 Define a new phoneme set

From the previous sections, we know the Chinese syllables are defined by the initial parts and the final parts, and each syllable has 5 possible tones. So, I decided to use the initial parts and the final parts as the phonetic labels. The tone information is combined directly with the final

parts. It means for a same final part with different tones become different phonemes. There are in total 220 phonemes for Chinese databases. The Appendix A give a whole description of the phoneme set used in the Chinese database

The phoneme definition consists of a name followed by eight features [Weeks 97]. The features are

```
vc      Vowel or consonant: + =vowel, consonant =-.
lng     Vowel length:  s =short,  l =long,  d =diphthong,  consonant =0.
h       Vowel height:  1 =high,   2 =mid,   3 =low,   consonant =-.
fr      Vowel frontness: 1 =front, 2 =mid,  3 =back,  consonant =-.
rnd     Lip rounding: + =rounded, - =not rounded.
typ     Consonant type:  s =stop, f =fricative, a =affricate, n =nasal, l =liquid, vowe
plc     Place of consonant articulation. l =labial, a =alveolar, p =palatal,
        b =labio-dental, d =dental, v =velar, vowel =0.
vox     Consonant voiced or unvoiced. + =voiced, - =unvoiced. vowel =-
```

### 3.3.2 Labelling

As mentioned in section 3.1, the two selected databases are already segmented and labelled. I have made a program "CorrectLab" to convert the old lab files to the new ones which use the new phoneme set.

This program take the old label files and the transcription file like input and give the new label files like output.
Use: `~ming/bin/CorrectLab speaker.txt OLD_LAB_DIR NEW_LAB_DIR`
The code source contains the following files:

1. "~ming/src/chlabel/CorrectLab.c"
2. "~ming/src/chlabel/adjustPy.c"
3. "~ming/src/chlabel/separatePy.c"

At first, I didn't want to label all the speech data again, so I made the first training of the database and test the resynthesis result, then I corrected the labels which have the problem in alignment. But when I made a new training, I found that there were still some other labels not well segmented. So, finally, I have checked and corrected all the speech data (two speakers). This work has consumed a great part of my work time.

### 3.3.3 Training the Database

The aim of training is to gather and characterize the files necessary for CHATR to build a synthesizer based on a speech corpus database. This work contains the following steps:

- extracting the prosodic features (power, pitch, pitch-marks etc)
- producing the unit descriptions

- building a binary representation of the full database index, pitch marks, acoustic parameters etc.
- making a CHATR representation of each utterance for testing the database with natural targets
- training some weights for unit selection

By now we have already

- sufficient speech material,
- corrected label files, and
- a new phoneme set

We need still other information for training the new database:

- the database description file: `db_description`
- the database parameters file (with information about the phonemeset, parameter weights, lexicon, intonation, duration parameters): `DBNAME_synth.ch`
- the training parameters file: `DBNAME_train.ch`

For full details of building a database from waveform files and phoneme label files, please refer to CHATR manual [Weeks 97] (section **creating & Training a Speech Synthesizer Database**).

Databases are large and therefore very likely to contain errors. Database errors are probably the most common cause of bad synthesis. Here I want to emphasize the most likely problems while I trained the Chinese database:

- *check the unit labels duration:* it exists some phonemes with very short durations which are not admitted by CHATR system
  use "`db_utils/check_labs`"
- check that the labels are all in the defined phoneme set:
  use "`/home/as62/ming/bin/Check_phoneset`"
- *check the location of labels within waveforms:*
  use "`/home/as62/ming/bin/Check_align`"
- *check the PhoneSets in the file DBNAME_synth.ch:* The phoneme set is very long, and the database may be not big enough to contain all the phonemes.

### 3.3.4 Testing a New Database

After training, the speech databases may be used as data for making new voices in CHATR. But at first, we must test the new database by using natural targets. The test is done by

- selecting the newly created database using the function:
  `(speaker_DBNAME)`
- testing the files generated during database creation and training using the function:
  `(Say (test_seg "file-id"))`

where `file-id` is a file-id from your newly created database. A synthesized version of the `file-id.wav` file will now be played. Of course ideally it should sound like the original! If not, we can use the function:
`(Save UnitLabels+ ''-'')` to see all the unit selections. If there are some errors in the selected units, we correct them and then make a new training.

Then we can test the database by synthesizing a new sentence, a new text etc. Once a database is proven to be stable it's `defspeaker` definition may be added to the file '`lib/data/itlspeakers.ch`' in the CHATR distribution so others may access it.

```
(defspeaker "cs9f" "/home/as62/ming/chinese/cs9f/db92/")
(defspeaker "cs4m" "/home/as62/ming/chinese/cs4m/db92/")
```

# Chapter 4

# Chinese synthesis within CHATR

## 4.1 Problems encountered in the unit selection

The strategy of unit selection is based on minimizing both distance between target segment and selected unit, and distance between selected unit and previous selected unit, i.e. the join (or the continuity distance). In theory, with an infinite database, the actual unit selection is not a problem, because there will be enough units to choose from in any given situation. But our actual database is not enough big to come close to this theoretical paradise. It happens quite often that there's no unit in the database that fulfills both the segmental and the prosodic requirements that have been predicted. It is why CHATR sometimes didn't choose the units we predicted by the context. This problem may result in a bad synthesis. I will show some examples of unit selection and present the problems encountered.

### 4.1.1 Initial part unit selection

The initial part units have the consonant features. For a same initial part, its characteristic change with that of the succeeded final part. The figure 4.1 show an example of the consonant unit selection.

To synthesize the syllable "qie1", we need the phoneme "q" and the phoneme "ie1". For the consonant phoneme "q", if it comes from the database "qin1", the result will be good because "in1" and "ie1" have the similar features, but if it comes from "qu1", the result will not be good, because "yv1" and "ie1" have very different features. To avoid this selection, we need to distinguish well these two units by giving a great phone distance.

### 4.1.2 Final part unit selection

The final part units have the vowel features. I have used the same phoneme to represent a final part when it is preceded by an initial part and when it is not. But for the final parts which begin by the letter "i", "u" or "ü", their pronunciations are little different in these two cases. So, in synthesis, it is better to distinguish these two cases, which can be recognized by the features of the preceded unit (vowel, consonant or pause signal). The figure 4.2 shows an

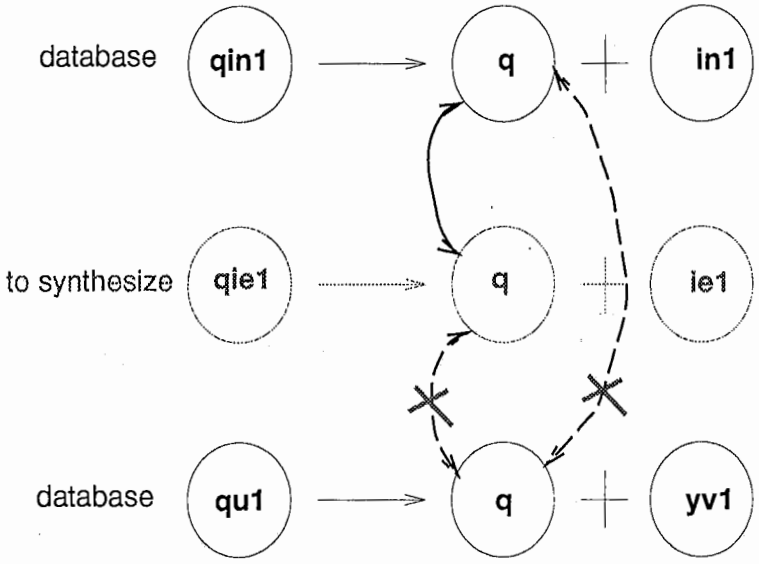


Figure 4.1: Initial part unit selection: example 1

example of the final part unit selection: the good selection and the bad selection.

A final part unit may be succeeded by a vowel, or by a consonant, or by a pause signal. As CHATR weighs continuity distance against unit distance, so it is possible for CHATR to select a vowel succeeded by a pause for a vowel succeeded by a consonant, shown in the figure 4.3, and this selection will produce a "break" between the vowel and the consonant in synthesis.

## 4.2 Improve the unit selection

In the last two sections, we have exposed the problems of unit selection encountered in Chinese synthesis. How to solve these problems ?

### 4.2.1 Calculation of phone distance

As our database is not big enough to give a unit in any case, we want to avoid some bad unit selections by giving an important distance between some very different units. The following is an extract of code source for calculating the phone distance between phones ap and bp.

```
if (ap == bp)
  return 0.0; /* same phone */
else if (ap->vowel != bp->vowel) /* vowel against consonant */
  return 1.5 ; /* 1.0 */
else if (ap->vowel == VOWEL) /* vowel cases */
  return 0.1 + (((ap->front != bp->front) ||
                 (ap->round != bp->round))*8 +
                (ap->height != bp->height)*3 +
```

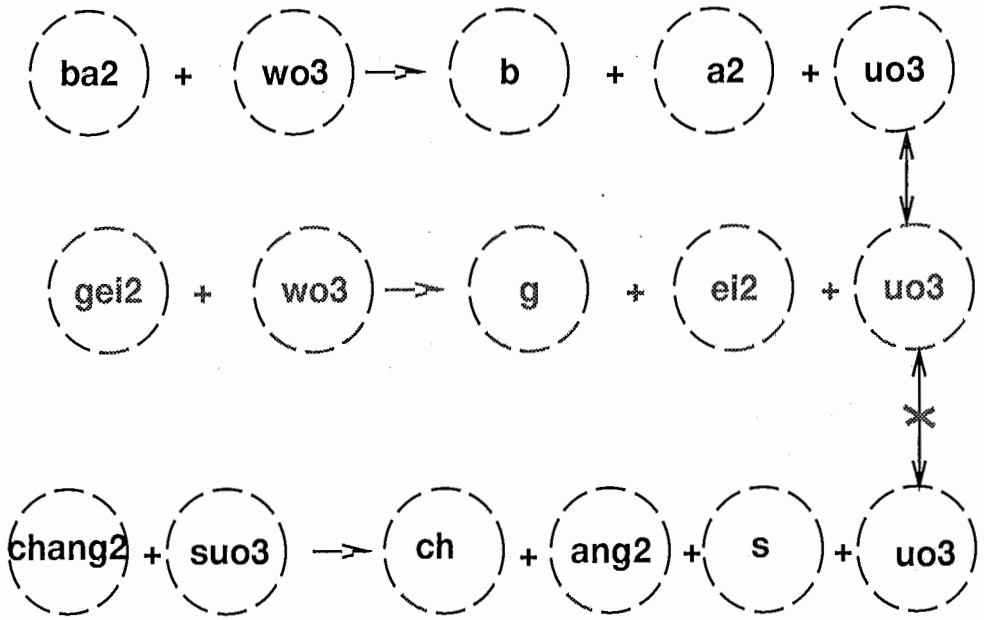


Figure 4.2: Final part unit selection: example 2

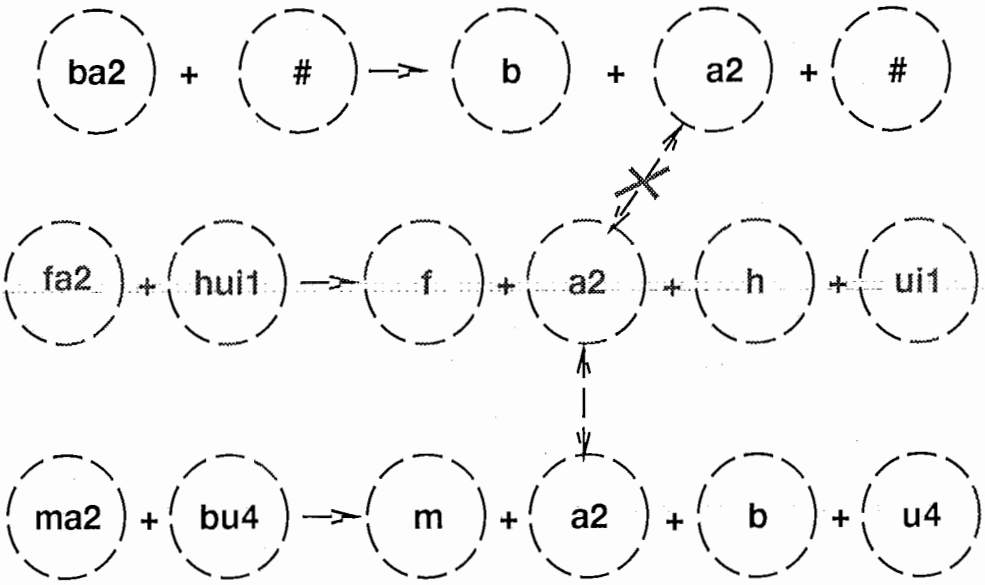


Figure 4.3: Final part unit selection: example 3

```
                (ap->length != bp->length)*1)/12.0;
  else {/* consonant cases */
    if (ap->place == 0 || bp->place == 0)
      return 1.5 ; /* consonant against pause */
    else
      return 0.1 + ((ap->c_type != bp->c_type)*3 +
                    (ap->place != bp->place)*4 +
                    (ap->voiced != bp->voiced)*1 +
                    (ap->round != bp->round)*2)/15.0 ;
  }
```

where we emphasize the features "round" and "front" for vowel cases to avoid the problem shown in the example 1; we give a great distance for 'vowel against consonant' cases to avoid the problem shown in the example 2; we give a great distance for 'pause against consonant' cases to avoid the problem shown in the example 3;

### 4.2.2 Units weights

The choice of weights also take an important role in the unit selection and can make a big difference to the quality of selection. After several tests, we have chosen the weight parameters as follows:

```
----  Weights used in selection ------
      (join_wt 1.0)
      (unit_wt 1.0)
      (lcontext_wt 1.0)
      (pp_phone_wt 1.0)
      (p_phone_wt  4.0)
      (n_phone_wt  12.0)
      (nn_phone_wt 1.0)
      (cand_thresh 0.9)
-------------------------------------
```

Another technique used is to exchange the values of `p_phone_wt` and `n_phone_wt` during building the candidate list. We think for the initial part units the succeeded unit is more important than the preceded unit, and opposite for the final part units.

```
source code: udb_generic.c
------
static void nus_find_best_path(Utterance utt)
{
    ---------
    max_phone_wt = nus_n_phone_weight ;
    min_phone_wt = nus_p_phone_weight ;
    if (nus_p_phone_weight > nus_n_phone_weight){
      max_phone_wt = nus_p_phone_weight ;
```

```
      min_phone_wt = nus_n_phone_weight ;
    }
    for (c=startc; SC_next(c) != SNIL; c=SC_next(c))
    {
      if (streq(current_lex->onfail,"CLTS")) {/* chinese database */
        ph = SC(c,NUSCand)->segtype ;
        if (!(udb_current->nus_phones[ph].phone->place)) {
          /* vowel or pause cases */
          nus_p_phone_weight = max_phone_wt ;
          nus_n_phone_weight = min_phone_wt ;
        }
        else {/* consonant cases */
          nus_p_phone_weight = min_phone_wt ;
          nus_n_phone_weight = max_phone_wt ;
        }
      }
    --------
}
-----
```

## 4.3 Improve the unit joint

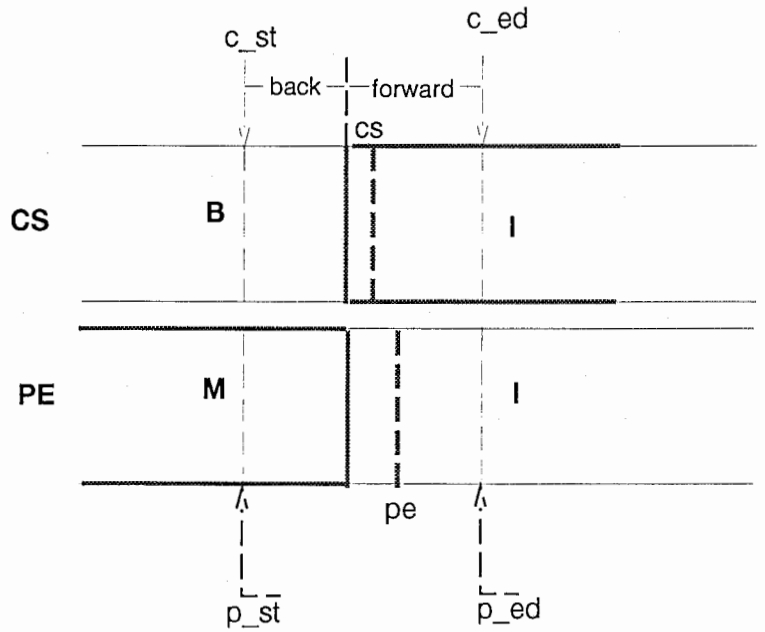


Figure 4.4: Looking for good places to join two units.

In the database, it is difficult to have a perfect alignment for each label. To take into account of this fact, CHATR system allows the endpoint of the previous unit and the start point of the current unit to move backward or forward a little bit (see figure 4.4). I will describe how

to determine the window size (backward and forward distance) and how to search for the best places to joint the two units. I will show also the difference between the general code sources and my personal code sources.

### 4.3.1 Window size determination

The window size determination is based on the duration of the correspondent units. But In the Chinese synthesis, only the forward searching is admitted when the previous units are not the same phones. Because the backward searching may include a part of the previous unit in the current unit.

```
General Code:
-------
  forward = f_dur/8;
  if (forward > MaxForwardDist) forward = MaxForwardDist;
  back = b_dur/4;
  if (back > MaxBackDist) back = MaxBackDist;
-------
Current Code:
------
  forward = f_dur/8;
  if (forward > MaxForwardDist) forward = MaxForwardDist;
  if (udb_unit_name(p_unit) == udb_unit_name(punit)) {
     back = b_dur/4;
     if (back > MaxBackDist) back = MaxBackDist;
   }
   else
     back = 0 ;
------
```

### 4.3.2 Search for the best places

In order to find good join points for units, CHATR system calculate the acoustic distance between a point in the current unit window and a point in the previous unit window, and search for the couple points which give the best score.

```
General Code:
for (cs = cand_start_win_st; cs < cand_start_win_ed; cs++) {
   int incr = cs - cand_start_win_st;
   c_vec = c_fvec[cs];    /* vector for current */
   for (pe = pcand_end_win_st + incr;
        pe<(pcand_end_win_st + incr+ 1) && pe<pcand_end_win_ed;
        pe++)
      p_vec = p_fvec[pe];  /* vector for previous */
Current Code:
for (cs = cand_start_win_st; cs < cand_start_win_ed; cs++) {
```

```
int incr = cs - cand_start_win_st;
c_vec = c_fvec[cs];    /* vector for current */
for (pe = pcand_end_win_st ; pe <pcand_end_win_ed;
     pe++)
   p_vec = p_fvec[pe];  /* vector for previous */
```

## 4.4 Chinese input within CHATR

The processing sequence of CHATR is as follows:

- Utterance Input,
- Phoneme Conversion,
- Prosody Prediction (duration and intonation),
- Unit Selection,
- Waveform Processing,
- Audio Output.

Input to CHATR is in the form of an utterance created by the **Utterance** command. Several types of input may be specified at quite different levels, varying from raw text to a simple waveform. Please refer to CHATR manual [Weeks 97] (see section **Basic Utterance Types**) for the detail descriptions of each possible input type.

In Chinese synthesis, we use usually two types of input:

- **Phoneme:** Input the utterance by giving the correspondent phonemes. Here we don't need the "Phoneme Conversion". But this method is not convenient, because the users have to know the correspondence between the words and the phonemes.
- **Text:** Input the utterance by giving the simple strings of words. Here we input the toned Pin-yin symbols. For the phoneme conversion, we don't use a lexicon, but we have added a piece of code source in the CHATR code source.

This code source named "`chinese.h`" is in the directory:
"~ming/src/chatr/src/include" and included in the file:
"~ming/src/chatr/src/lex/lexicon.c"

```
source code: lexicon.c
-----
#include "chinese.h"
List lex_lookup(char *text,List features)
{
  -----
  else if (streq(current_lex->onfail,"CLTS")) {/* chinese database */
```

```
    if (!strcmp(actionfrom,"Text")) {
      strcpy (text0, text) ;
      transform_text_to_phonemes (text0, text) ;
    }
    entry = mlts_lookup(text,features,"CLTS");
  }
  -----
}
-----
```

It exits also an executable program named "py2phoneme" which convert the Pin-yin symbols to phonemes. The code source is in the path:

"~ming/src/chlabel/py2phoneme.c"

# Chapter 5

# Chinese Lexicon

We know the written Chinese is not phonetic but ideographical. The recorded documents are always in han-zi, so we need a lexicon by which we can convert the Chinese orthography to the toned Pin-yin symbols.

We have got a mandarin lexicon from CALLHOME databases. This lexicon contains 44484 words, and it contains the following information:

- Chinese characters and words in **GB** code,
- the toned Pin-yin symbols,
- the realized tones of characters in the words,
- and the word categories(noun, verb, adjective...) etc.

For this, I have developed a program called "`gblex`" which convert the Chinese orthography in **GB** code to the toned pin-yin symbols:

Use: `gblex input.gb output.py`

The principle of this program is as follows:

- organize the lexicon for efficient access
- divide the input text in phrases
- look for the longest words (the pronunciation of a word is fixed)
- get the toned Pin-yin symbols if word found
- and give the realized tones if tones change
- or give the more frequent pronunciation if no word found

The program "`gblex`" need the following data:

- the CALLHOME DATABASE mandarin lexicon:
  "~ming/ch_softw/lex/ma_lex.v03"

- the more frequent pronunciation lexicon:
  "~ming/ch_softw/gb2Py/mulpin.gb"

- the punctuation lexicon:
  "~ming/ch_softw/gb2Py/bdfh.gb"

The lexicons "`mulpin.gb`" and "`bdfh.gb`" are created by the author. The source code is in the path:

"~ming/ch_softw/gb2py/gblex.c"

# Chapter 6

# Remaining problems

Here I will discuss some remaining problems. First the database is not big enough. We have found that there are only one or two samples for some units. So during synthesis, for these units it is possible neither the segmental nor the prosodic requirements are satisfactory. If a such unit is selected, the result of synthesis should be bad. So we need a bigger database.

Second, in our Chinese synthesis, because of time limitations, we have not considered the problems about prosody prediction. Actually, for duration prediction, we have used "Average+" method which average phone duration with some simple alterations over syllables available for stressed/accented and phrase final. But in Chinese, the duration of a character (syllable) is dependent on the context. Sometimes, Chinese word consists of more than one characters, so duration prediction must be carried out on the word, not the character. For the intonation prediction, we use actually a Japanese (JToBI) Linear Regression model (`chatr/lib/data/mht_lrf0.ch`). Of course, Japanese and Chinese intonation are different. So, for the text input synthesis, especially for a long text synthesis, we have found that the intonations are not natural. To improve the prosody prediction, we may need Chinese ToBI labelling.

Finally, for the conversion of Chinese orthography to Pin-yin symbols, we give the more frequent pronunciation when no word is found. This solution contains errors. For some Chinese characters, their pronunciation depend on the context, so for determining the pronunciation, we need to

- analyze the structure of the phrase and the context-dependent word category (noun, verb, adjective etc.)
- make rules for the special multi-pronunciation characters

This work is also useful for the prosody prediction.

# Chapter 7

# Conclusion

During the six months at ITL, I have realized the following works:

- **creating a CHATR chinese database:** Two databases have been built (a male and a female).
- **improvement of CHATR for Chinese speech synthesis** in
  - unit selections
  - unit joints
- **automatic conversion from Chinese orthography to Pin-yin symbols:** which can be input directly in CHATR system.

It is clear that a good database on segmental level is very important for having a good synthesis results. Also, the improvement in the unit selection and the unit joint make the synthesis voices more intelligible. To have better synthesis results, we need improve the prosody prediction. A bigger database which contains more units and more variety in prosodic patterns is necessary.

# Bibliography

[Weeks 97] Martyn Weeks. *The CHATR User Guide.* http://www.itl. atr.co.jp/~mweeks/. ATR Interpreting Telecommunications Research Laboratories, 1997.

[HKU96] Y.Q.Zu, W.X.Li, M.C.Ho, C.Chan *HKU96 – A Mandarin Corpus CD-ROM Version.* Speech Lab. Dept. of Computer Science Univ. of Hong Kong, 1996

# Appendix A

# The new phoneme set definition

```
(Phoneme Def chinese
 ;name vc lng h fr rnd typ plc vox
 (
  (b - 0 - - - s l -)
  (p - 0 - - - s l -)
  (m - 0 - - - n l -)
  (f - 0 - - - f b -)
  (d - 0 - - - s a -)
  (t - 0 - - - s a -)
  (n - 0 - - - n a -)
  (l - 0 - - - l a -)
  (g - 0 - - - s v -)
  (k - 0 - - - s v -)
  (h - 0 - - - f v -)
  (j - 0 - - - a p -)
  (q - 0 - - - a p -)
  (x - 0 - - - f p -)
  (zh - 0 - - + a p -)
  (ch - 0 - - + a p -)
  (sh - 0 - - + f p -)
  (r - 0 - - + l p +)
  (z - 0 - - - f d -)
  (c - 0 - - - f d -)
  (s - 0 - - - f d -)
  (a0 + s 3 2 - 0 0 -)
  (a1 + s 3 2 - 0 0 -)
  (a2 + s 3 2 - 0 0 -)
  (a3 + s 3 2 - 0 0 -)
  (a4 + s 3 2 - 0 0 -)
  (ang0 + s 3 2 - 0 0 -)
  (ang1 + s 3 2 - 0 0 -)
  (ang2 + s 3 2 - 0 0 -)
  (ang3 + s 3 2 - 0 0 -)
```

```
(ang4 + s 3 2 - 0 0 -)
(an0 + s 2 2 - 0 0 -)
(an1 + s 2 2 - 0 0 -)
(an2 + s 2 2 - 0 0 -)
(an3 + s 2 2 - 0 0 -)
(an4 + s 2 2 - 0 0 -)
(ao0 + d 3 2 - 0 0 -)
(ao1 + d 3 2 - 0 0 -)
(ao2 + d 3 2 - 0 0 -)
(ao3 + d 3 2 - 0 0 -)
(ao4 + d 3 2 - 0 0 -)
(ai0 + d 2 2 - 0 0 -)
(ai1 + d 2 2 - 0 0 -)
(ai2 + d 2 2 - 0 0 -)
(ai3 + d 2 2 - 0 0 -)
(ai4 + d 2 2 - 0 0 -)
(e0 + s 1 2 - 0 0 -)
(e1 + s 1 2 - 0 0 -)
(e2 + s 1 2 - 0 0 -)
(e3 + s 1 2 - 0 0 -)
(e4 + s 1 2 - 0 0 -)
(en0 + s 1 2 - 0 0 -)
(en1 + s 1 2 - 0 0 -)
(en2 + s 1 2 - 0 0 -)
(en3 + s 1 2 - 0 0 -)
(en4 + s 1 2 - 0 0 -)
(eng0 + s 1 2 - 0 0 -)
(eng1 + s 1 2 - 0 0 -)
(eng2 + s 1 2 - 0 0 -)
(eng3 + s 1 2 - 0 0 -)
(eng4 + s 1 2 - 0 0 -)
(ei0 + d 1 2 - 0 0 -)
(ei1 + d 1 2 - 0 0 -)
(ei2 + d 1 2 - 0 0 -)
(ei3 + d 1 2 - 0 0 -)
(ei4 + d 1 2 - 0 0 -)
(er0 + s 1 2 + 0 0 -)
(er1 + s 1 2 + 0 0 -)
(er2 + s 1 2 + 0 0 -)
(er3 + s 1 2 + 0 0 -)
(er4 + s 1 2 + 0 0 -)
(o0 + s 2 3 - 0 0 -)
(o1 + s 2 3 - 0 0 -)
(o2 + s 2 3 - 0 0 -)
(o3 + s 2 3 - 0 0 -)
(o4 + s 2 3 - 0 0 -)
(ong0 + s 2 3 - 0 0 -)
```

```
(ong1 + s 2 3 - 0 0 -)
(ong2 + s 2 3 - 0 0 -)
(ong3 + s 2 3 - 0 0 -)
(ong4 + s 2 3 - 0 0 -)
(ou0 + d 2 3 - 0 0 -)
(ou1 + d 2 3 - 0 0 -)
(ou2 + d 2 3 - 0 0 -)
(ou3 + d 2 3 - 0 0 -)
(ou4 + d 2 3 - 0 0 -)
(<c>i0 + s 1 1 - 0 0 -)
(<c>i1 + s 1 1 - 0 0 -)
(<c>i2 + s 1 1 - 0 0 -)
(<c>i3 + s 1 1 - 0 0 -)
(<c>i4 + s 1 1 - 0 0 -)
(<ch>i0 + s 1 1 + 0 0 -)
(<ch>i1 + s 1 1 + 0 0 -)
(<ch>i2 + s 1 1 + 0 0 -)
(<ch>i3 + s 1 1 + 0 0 -)
(<ch>i4 + s 1 1 + 0 0 -)
(i0 + s 1 1 - 0 0 -)
(i1 + s 1 1 - 0 0 -)
(i2 + s 1 1 - 0 0 -)
(i3 + s 1 1 - 0 0 -)
(i4 + s 1 1 - 0 0 -)
(in0 + s 1 1 - 0 0 -)
(in1 + s 1 1 - 0 0 -)
(in2 + s 1 1 - 0 0 -)
(in3 + s 1 1 - 0 0 -)
(in4 + s 1 1 - 0 0 -)
(ing0 + s 1 1 - 0 0 -)
(ing1 + s 1 1 - 0 0 -)
(ing2 + s 1 1 - 0 0 -)
(ing3 + s 1 1 - 0 0 -)
(ing4 + s 1 1 - 0 0 -)
(ie0 + d 1 1 - 0 0 -)
(ie1 + d 1 1 - 0 0 -)
(ie2 + d 1 1 - 0 0 -)
(ie3 + d 1 1 - 0 0 -)
(ie4 + d 1 1 - 0 0 -)
(ia0 + d 2 1 - 0 0 -)
(ia1 + d 2 1 - 0 0 -)
(ia2 + d 2 1 - 0 0 -)
(ia3 + d 2 1 - 0 0 -)
(ia4 + d 2 1 - 0 0 -)
(iao0 + d 2 1 - 0 0 -)
(iao1 + d 2 1 - 0 0 -)
(iao2 + d 2 1 - 0 0 -)
```

```
(iao3 + d 2 1 - 0 0 -)
(iao4 + d 2 1 - 0 0 -)
(ian0 + d 2 1 - 0 0 -)
(ian1 + d 2 1 - 0 0 -)
(ian2 + d 2 1 - 0 0 -)
(ian3 + d 2 1 - 0 0 -)
(ian4 + d 2 1 - 0 0 -)
(iang0 + d 2 1 - 0 0 -)
(iang1 + d 2 1 - 0 0 -)
(iang2 + d 2 1 - 0 0 -)
(iang3 + d 2 1 - 0 0 -)
(iang4 + d 2 1 - 0 0 -)
(iou0 + l 2 1 + 0 0 -)
(iou1 + l 2 1 + 0 0 -)
(iou2 + l 2 1 + 0 0 -)
(iou3 + l 2 1 + 0 0 -)
(iou4 + l 2 1 + 0 0 -)
(iong0 + l 2 1 + 0 0 -)
(iong1 + l 2 1 + 0 0 -)
(iong2 + l 2 1 + 0 0 -)
(iong3 + l 2 1 + 0 0 -)
(iong4 + l 2 1 + 0 0 -)
(u0 + s 1 3 + 0 0 -)
(u1 + s 1 3 + 0 0 -)
(u2 + s 1 3 + 0 0 -)
(u3 + s 1 3 + 0 0 -)
(u4 + s 1 3 + 0 0 -)
(ua0 + d 2 3 + 0 0 -)
(ua1 + d 2 3 + 0 0 -)
(ua2 + d 2 3 + 0 0 -)
(ua3 + d 2 3 + 0 0 -)
(ua4 + d 2 3 + 0 0 -)
(uai0 + d 2 3 + 0 0 -)
(uai1 + d 2 3 + 0 0 -)
(uai2 + d 2 3 + 0 0 -)
(uai3 + d 2 3 + 0 0 -)
(uai4 + d 2 3 + 0 0 -)
(uan0 + d 2 3 + 0 0 -)
(uan1 + d 2 3 + 0 0 -)
(uan2 + d 2 3 + 0 0 -)
(uan3 + d 2 3 + 0 0 -)
(uan4 + d 2 3 + 0 0 -)
(uang0 + d 2 3 + 0 0 -)
(uang1 + d 2 3 + 0 0 -)
(uang2 + d 2 3 + 0 0 -)
(uang3 + d 2 3 + 0 0 -)
(uang4 + d 2 3 + 0 0 -)
```

```
(uen0 + d 1 3 + 0 0 -)
(uen1 + d 1 3 + 0 0 -)
(uen2 + d 1 3 + 0 0 -)
(uen3 + d 1 3 + 0 0 -)
(uen4 + d 1 3 + 0 0 -)
(uei0 + d 1 3 + 0 0 -)
(uei1 + d 1 3 + 0 0 -)
(uei2 + d 1 3 + 0 0 -)
(uei3 + d 1 3 + 0 0 -)
(uei4 + d 1 3 + 0 0 -)
(ueng0 + d 1 3 + 0 0 -)
(ueng1 + d 1 3 + 0 0 -)
(ueng3 + d 1 3 + 0 0 -)
(ueng4 + d 1 3 + 0 0 -)
(uo0 + d 1 3 + 0 0 -)
(uo1 + d 1 3 + 0 0 -)
(uo2 + d 1 3 + 0 0 -)
(uo3 + d 1 3 + 0 0 -)
(uo4 + d 1 3 + 0 0 -)
(yv0 + s 1 3 - 0 0 -)
(yv1 + s 1 3 - 0 0 -)
(yv2 + s 1 3 - 0 0 -)
(yv3 + s 1 3 - 0 0 -)
(yv4 + s 1 3 - 0 0 -)
(yvn0 + d 1 3 - 0 0 -)
(yvn1 + d 1 3 - 0 0 -)
(yvn2 + d 1 3 - 0 0 -)
(yvn3 + d 1 3 - 0 0 -)
(yvn4 + d 1 3 - 0 0 -)
(yve0 + d 1 3 - 0 0 -)
(yve1 + d 1 3 - 0 0 -)
(yve2 + d 1 3 - 0 0 -)
(yve3 + d 1 3 - 0 0 -)
(yve4 + d 1 3 - 0 0 -)
(yvan0 + d 2 3 - 0 0 -)
(yvan1 + d 2 3 - 0 0 -)
(yvan2 + d 2 3 - 0 0 -)
(yvan3 + d 2 3 - 0 0 -)
(yvan4 + d 2 3 - 0 0 -)
(ng2 + s 1 2 - 0 0 -)
(ng3 + s 1 2 - 0 0 -)
(ng4 + s 1 2 - 0 0 -)
(# - 0 - - - 0 0 -)
(P0 - 0 - - - 0 0 +)
(P1 - 0 - - - 0 0 +)
(P2 - 0 - - - 0 0 +)
(P3 - 0 - - - 0 0 +)
```

```
(X - 0 - - - 0 0 +)
(Xc - 0 - - - 0 0 +)
(Xv + 0 - - - 0 0 +) ))
```

# Appendix B

# Collecting the HKU CD-ROM database

The HKU CD-ROM database is stored in five CD-ROMs. This database contains 20 native Mandarin speakers (10 males and 10 females). For the contents of each CD-ROM, please refer to [?].

When a speaker is selected, we do the lecture of the correspondent CD-ROM in a PC, and get three types of files:

- transcription file "speaker.txt"
- lab file "speaker.lab"
- all waveform files correspond to the speaker

Then we can use "`ftp`" to transfer all the files to our work station under a new created directory for the speaker. we put all the waveform files in the directory "`./wav`"

Then we need to do several treatments about the waveform files:

1. move to the waveform directory and swap the waveform files:
   use `~ming/bin/swap.sh`
2. lower case the waveform file names:
   use `~ming/bin/tolower.sh`

After have done this operation, we can check the waveform files by using "`xwaves`" or using "`naplay`".

Now, we need to separate the label file for each utterance. to do that:
use "`~ming/bin/getlabf speaker.lab`" This operation will give us a directory "`lab/`" with the all separate label files in a format which is useful to CHATR.

# Appendix C

# Listing of the programs: CorrectLab py2phoneme gblex

```
/*****************************************************************************/
/*  Input : *.txt *.lab                                                      */
/*  The *.txt file :                                                         */
/*       is the transcription file consisting of all the orthographic        */
/*  transcriptions of the same speaker, stored record by record. each record */
/*  correspons to the transcription of a waveform. Each record contains 5 lines.  */
/*  Each line is ended with a newline symbol. The first line of record is the */
/*  file name of the corresponding utterance waveform file. The second line is a  */
/*  line of symbol '-' which is simply a separator. The third line is the ortho-  */
/*  graphic transcription of the record Mandarin utterance. the forth line is the */
/*  corresponding Pinyin symbols with tones. The last lines of each record is */
/*  line of symbol '*' which indicates the end of the record. Then , the next */
/*  starts another record, and that repeats until the end of file.           */
/*                                                                           */
/*  The label files :                                                        */
/*       consist of all the phonetic labels of utterances spoken by the same */
/*  speaker, generateds by Hong Kong University. For each chinese character, */
/*  there is only one syllable corresponding one Pinyin. A pinyin consist of the  */
/*  initial part containig only the consonants, and the final part containing the */
/*  vowels and the consonants "n" and "g". In these label files, the final parts  */
/*  are divided into the phone units, these is not the tone information.     */
/*                                                                           */
/*  Output :                                                                 */
/*       This  program is used to generate the new label files. We use the initial*/
/*  part and the final part combined with the tone as the label units. We take */
/*  from the *.txt file the pinyin symbols with tones and from the lab files the  */
/*  the phonetic labels's ending position.                                   */
/*  the file "error.txt" contain all the label files they could have the errors */
/*  coming from the original lab files. Please check them before using.       */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <string.h>
#include "def.h"

void main (int argc, char **argv)
[
  register i, j, l ;
  char c, words[10] ;
  int nbchar, nbstr, okc, okd, nblines, lensymb, lwd, lp ;
  int ltest, nb_l, nn, nbfile, oldgroupe ;
  FILE *pf, *pflab, *fopen (), *pferr ;
  char nametxt[80], strs[30], newlab[80] ;
  char namelab[80], testchar[30], StrCurrent[5] ;
  char txtline[MaxLineSize], labline[MaxLineSize] ;
  char symb[MaxNblines][10] ;
  char Phonemes[MaxNblines][10] ;
  float times[MaxNblines], val_t, delt, coef ;
  int color[MaxNblines], WdIndex[MaxNblines] ;
  char WdAccent[MaxNbWords] ;
  char substr[5][5], LABDIR[30], NEWLABDIR[30] ;
  int nsub, nlast ;

  if (argc < 4) [
    fprintf (stderr, "Usage : %s name.txt LABDIR NEWLABDIR\n", argv[0]) ;
    printf ("input the name of file *.txt : ") ;
    scanf ("%s",nametxt) ; /* get the name by standard input */
    printf ("input the lab directory and the newlab durectory : ") ;
    scanf ("%s %s", LABDIR, NEWLABDIR) ;
  ]
  else [
    strcpy (nametxt, argv[1]) ; /* get the name from commande line */
    strcpy (LABDIR, argv[2]) ;
    strcpy (NEWLABDIR,argv[3]) ;
  ]

  printf ("nametxt: %s\n", nametxt) ;

  pferr = fopen ("error.txt", "w") ; /* open a file for saving all the
                                        errors found in the files *.lab */
  nbfile = 0 ;
  pf = fopen (nametxt, "r") ; /* open the file */
  if (pf == NULL) [
    printf ("Error : cannot open file %s \n", nametxt) ;
    exit (-1) ;
  ]
```

```
while ( fgets (txtline, 300, pf)!= NULL) {/* read the name of the corresponding
                                             waveform of the current record */
  if (txtline[0] != 'c') break ; /* we get only the sentence record */
  sscanf (txtline, "%s", strs) ;
  strcpy (testchar, strchr(strs,'.')) ;
  ltest = strlen(strs) - strlen(testchar) ;
  strcpy (namelab, LABDIR) ;
  strncat (namelab, strs, ltest) ;
  strcat (namelab, ".lab") ;

  printf ("%s\n", namelab) ;

  /* reading the label file */
  pflab = fopen (namelab, "r") ;
  if (pflab == NULL) {
    printf ("Error : cannot open file %s \n",namelab);
    exit (-1) ;
  }

  do {
    fgets (labline, 300, pflab) ;
  }while (labline[0] != '#') ; /* the symbol "#" indicate the beginning
                                  of the label data */

  nblines = 0 ;
  while (fgets (labline, 300, pflab) != NULL) {
    sscanf (labline, "%f %d %s",
            &times[nblines], &color[nblines], symb[nblines]) ;
    if (nblines > 0) {
      delt = times[nblines] - times[nblines-1] ;
      if (delt <= MinDur)
        fprintf (pferr,"%s :the duration of the label \"%s\" is not good \n",
                 namelab, symb[nblines]) ;
    }
    lensymb = strlen (symb[nblines]) ;

    /* remove the characters between "(" and ")" from the current symbol */
    if (strchr (symb[nblines], OPENCHAR) != NULL) {
      strcpy (strs, "\0") ;
      strcpy (testchar, strchr (symb[nblines], OPENCHAR)) ;
      ltest = lensymb - strlen(testchar) ;
      if (ltest > 0) {
        strncpy (strs, symb[nblines], ltest);
        strs[ltest] = '\0' ;
      }
      strcpy (testchar, strchr(symb[nblines],CLOSECHAR)) ;
      ltest = strlen (testchar) ;
      if (ltest > 1)
        strcat(strs,testchar+1);
      strcpy (symb[nblines], strs) ;
    }

    nblines++ ;
  }
  fclose (pflab) ; /* end reading the label file */

  fgets (txtline, 300, pf) ; /* the second line "-------"*/
  fgets (txtline, 300, pf) ; /* the third line : chinese orthographic transcriptions */
  fgets (txtline, 300, pf) ; /* the fourth line : line of chinese text in Pinyin*/

  nbchar = strlen (txtline) ;
  nbstr = 0 ;
  strcpy (strs,"\0") ;
  okc = 0 ;
  lp = 0 ;
  i = 0 ;
  while (i<nbchar) {
    c = txtline[i] ;
    okd = isdigit(c) ;
    if (c != ' ' && c != '\n' && !okd) {
      okc++ ;
      strncat (strs, &c, 1) ;
      i++ ;
    }
    else if (okd && okc > 0) {
      i++ ;
      okc = 0 ;
      lwd = strlen (strs) ;
      if (lwd > 6) {/* the length of a pingyin can't depasse 6 */
```

```
          printf ("Pinyin error or tone missing at line %d\n", nbfile*5+4) ;
          exit (-1) ;
        }
        if (c == '5') c = '0' ; /* tone 5 is the same as tone 0 */
        WdAccent[nbstr] = c ;
        adjustPinYin (strs, lwd, words) ;
        SeparateInitialFinal (words, &lp, Phonemes, WdIndex) ;
        nbstr++ ;
        strcpy (strs, "\0") ;
      }
      else
        i++ ;
    }
    fgets (txtline, 300, pf) ; /* the end line  of the current record */

    /* generate the new label file */
    strcpy (newlab, NEWLABDIR) ;

    while (strchr (namelab, '/') != NULL) {
      strcpy (testchar, (strchr (namelab, '/')+1)) ;
      strcpy (namelab, testchar) ;
    }
    strcat (newlab,namelab) ;

    printf ("newlab: %s\n", newlab) ;
    pflab = fopen (newlab, "w") ;
    if (pflab == NULL) {
      printf ("Error : cannot open file %s \n",newlab);
      exit (-1) ;
    }

    fprintf (pflab, "#\n") ;
    fprintf (pflab, "%f %d %s\n", times[0], color[0], symb[0]) ;

    nn = 1 ;
    nb_l = nblines -1 ;
    nbstr = 0 ;
    for (l=0; l<lp; l++) {
      if (nn == nb_l) {
        fprintf (pferr,"%s: has lost the symbol \"%s\" at line %d or the Pinyin is wrong\n", namelab
, Phonemes[l],(nn+2)) ;
        break ;
      }

      while (!strcmp(symb[nn],"#")){/* here # means a silence */
        fprintf (pflab, "%f %d %s\n", times[nn], color[nn], symb[nn]) ;
        nn++ ;
      }

      strcpy (StrCurrent, Phonemes[l]) ;

      if (!strcmp (Phonemes[l],"<c>i") || !strcmp (Phonemes[l],"<ch>i"))
        strcpy (Phonemes[l], "i") ;

      if (WdIndex[l] == 1){
        strncat (StrCurrent, &WdAccent[nbstr],1) ;
        nbstr++;
      }
      else if (WdIndex[l] == 2)
        strncat (StrCurrent, &WdAccent[nbstr-1],1) ;

      if (!strcmp (symb[nn], Phonemes[l])) {
        fprintf (pflab, "%f %d %s\n", times[nn], color[nn], StrCurrent) ;
        nn++ ;
      }
      else {
        if (!WdIndex[l]) {
          fprintf (pferr,"%s: the label symbol \"%s\" could be lost at line %d\n",
                   namelab, Phonemes[l],(nn+2)) ;
          continue ;
        }
        else {
          nlast = nn ;
          ltest = strlen (Phonemes[l]) - strlen(symb[nn]);
          if (ltest < 0) {
            fprintf (pferr, "%s : the label symbol \"%s\" could be lost at line %d\n",
                     namelab, Phonemes[l],(nn+2)) ;
            continue ;
          }
```

```
          strcpy (testchar, symb[nn]) ;
          if (ltest > 0) {
            j = 0 ;
            while (j < ltest) {
              if (!strcmp (symb[nn+1], "#")) {
                fprintf (pferr,"%s : has lost some letters at line %d before \"#\"\n",
                         namelab, nn+3) ;
                break ;
              }
              else {
                if ((j+strlen(symb[nn+1])) <= ltest){
                  nn++ ;
                  j += strlen(symb[nn]) ;
                  strcat (testchar,symb[nn]) ;
                  nlast = nn ;
                  if (nn >= nb_1) {
                    fprintf (pferr, "%s: has lost some letters at line %d\n",
                             namelab, (nn+2)) ;
                    break ;
                  }
                }
                else {
                  fprintf (pferr, "%s: has lost some letters at line %d\n",
                           namelab, (nn+2)) ;
                  break ;
                }
              }
            }
          }

          fprintf(pflab, "%f %d %s\n",times[nlast],color[nlast],StrCurrent);
          nn++ ;
        }
      }
    }

    if (strcmp (symb[nb_1], "#") != 0) {
      fprintf (pferr, "%s : has lost the last line!!\n", namelab) ;
      exit (-1) ;
    }
    fprintf (pflab, "%f %d #\n", times[nb_1], color[nb_1]) ;
    fclose (pflab) ;

    nbfile++ ;
#ifdef TEST
    if (nbfile > 1) break ;
#endif
  }
  fclose (pferr) ;
  fclose (pf) ;

}
```

```
#include <string.h>
#include "def.h"

void adjustPinYin (char *strs, int lwd, char *strsnew) ;
static void adjustment_y (char *strs1, int lwd, char *strs2) ;
static void adjustment_u (char *strs1, int lwd, char *strs2) ;
static void adjustment_w (char *strs1, char *strs2) ;

/* to adjust the PinYin so that it will be composed of
   standards consonant's part and vowel's part */

void adjustPinYin (char *strs, int lwd, char *strsnew)
{
  char firstc, lastc, c_R, res[10] ;
  int i ;

  c_R = 'r' ;
  firstc = strs[0] ;
  strsnew[0] = strs[0] ;
  strsnew[1] = '\0' ;
  strcpy (res, "\0") ;
  if (lwd >= 3) {
    for (i=1; i<lwd-2; i++)
      strncat (strsnew, &strs[i],1) ;
    for (i=2; i>0; i--)
      strncat(res,&strs[lwd-i], 1) ;
  }

  if (!strcmp (res, "iu"))
    strcat (strsnew, "iou") ;
  else if (!strcmp(res,"ui"))
    strcat (strsnew, "uei") ;
  else if (!strcmp (res,"un") && firstc != 'y' && firstc != 'x' && firstc != 'q' && firstc != 'j')
    strcat (strsnew, "uen") ;
  else if (!strcmp (res,"ue") && firstc != 'y' && firstc != 'x' && firstc != 'q' && firstc != 'j')
    strcat (strsnew, "yve") ;
  else if (!strcmp(res, "uu"))
    strcat (strsnew, "yv") ;
  else
    strcpy (strsnew, strs) ;

  lwd = strlen (strsnew) ;
  strcpy (res, strsnew) ;

  if (res[lwd-1] == 'r' && lwd > 2) {
    if ((res[lwd-2] != 'e') || IsVowel(res[lwd-3])){
      res[lwd-1] = 'e' ;
      strncat (res, &c_R, 1) ;
    }
  }

  switch (firstc) {
  case 'y' : adjustment_y (res, lwd, strsnew) ;
    break ;
  case 'w' : adjustment_w (res, strsnew) ;
    break ;
  case 'j' : adjustment_u (res, lwd, strsnew) ;
    break ;
  case 'q' : adjustment_u (res, lwd, strsnew) ;
    break ;
  case 'x' : adjustment_u (res, lwd, strsnew) ;
    break ;
  default : strcpy (strsnew, res) ;
    break ;
  }

}

/* when the first character of Pinyin is "y", we do the following adjustment */
static void adjustment_y (char *strs1, int lwd, char *strs2)
{
  if (strs1[1] == 'i') {/* here we remove "y" which is not pronounced */
    strcpy(strs2, strchr (strs1, 'i')) ;
  }
  else if (strs1[1] == 'u') {/* here we remove "y" and replace "u" by "yv" */
    strcpy (strs2, "yv");
    if (lwd > 2) strcat (strs2, (strchr (strs1,'u')+1)) ;
  }
  else {/* here "y" is pronounced like "i", we replace it */
```

```
    strcpy (strs2, strs1) ;
    strs2[0] = 'i' ;
  }

}

/* when the first character of Pinyin is "w", we do the following adjustment */
static void adjustment_w (char *strs1, char *strs2)
{
  if (strs1[1] == 'u') /* here we remove "w" which is not pronounced */
    strcpy (strs2,strchr (strs1, 'u')) ;
  else { /* "w" is pronounced like "u", we repace it */
    strcpy (strs2, strs1) ;
    strs2[0] = 'u' ;
  }

}

/* when the first character of Pinyin is "x" or "q" or "j" and the second
   is "u",  we do the following adjustment */
static void adjustment_u (char *strs1, int lwd, char *strs2)
{
  if (strs1[1] == 'u') { /* here "u" is pronounced like "yv", we repace it */
    strs2[0] = strs1[0] ;
    strs2[1] = '\0' ;
    strcat (strs2, "yv") ;
    if (lwd > 2) strcat (strs2, (strchr (strs1,'u')+1)) ;

  }
  else
    strcpy (strs2, strs1) ;

}
```

```
#include "def.h"

/* decompose the symbol in vowels and consonants. */
void SeparateInitialFinal (char *words,
                           int *lp,
                           char Phonemes[MaxNblines][10],
                           int *WdIndex) ;

int IsVowel (char c) ;
static int IsGroupCSZ (char *strs);
static int IsRetroflex (char *strs) ;

/* divide Pinyin into two parts: the initial and the final. */
void SeparateInitialFinal (char *words,
                           int *lp,
                           char Phonemes[MaxNblines][10],
                           int *WdIndex)
{
  int nb, ok, len, i, j ;
  char strs[10] ;

  nb = *lp ;

  len = strlen (words) ;

  ok = IsVowel (words[0]);
  if (!ok) {
    Phonemes[nb][0] = words[0] ;
    Phonemes[nb][1] = '\0' ;

    WdIndex[nb] = ok ;

    for (i=1; i<len; i++) {
      ok = IsVowel (words[i]);
      if (!ok)
        strncat (Phonemes[nb], &words[i],1) ;
      else {
        nb++ ;
        strcpy (strs,"\0") ;
        for (j=i;j<len; j++)
          strncat (strs, &words[j],1) ;
        break ;
      }
    }
  }
  else
    strcpy (strs,words) ;

  len = strlen (strs) ;

  if (strs[len-1] == 'r' && len > 2) {
    strncpy (Phonemes[nb], strs, len-2) ;
    Phonemes[nb][len-2] = '\0' ;
    if (!strcmp(Phonemes[nb], "i")) {
      if (IsGroupCSZ (words))
        strcpy (Phonemes[nb], "<c>i") ;
      else if (IsRetroflex (words))
        strcpy (Phonemes[nb], "<ch>i") ;
    }

    WdIndex[nb] = 1 ;
    nb++ ;
    strcpy (Phonemes[nb],"er") ;
    WdIndex[nb] = 2 ;
    nb++ ;
  }
  else {
    strcpy (Phonemes[nb], strs) ;
    if (!strcmp(Phonemes[nb], "i")) {
      if (IsGroupCSZ (words))
        strcpy (Phonemes[nb], "<c>i") ;
      else if (IsRetroflex (words))
        strcpy (Phonemes[nb], "<ch>i") ;
    }
    WdIndex[nb] = 1 ;
    nb++ ;
  }

  *lp = nb ;
```

```
}

static int IsGroupCSZ (char *strs)
{
  int ok=0, len ;

  len = strlen (strs) ;
  if (len == 2 && (strs[0] == 'c' || strs[0] == 's' || strs[0] == 'z'))
    ok = 1 ;
  return (ok) ;
}

static int IsRetroflex (char *strs)
{
  int ok=0, len ;

  len = strlen (strs) ;
  if (strs[0] == 'r' || (len == 3 && (strs[0] == 'c' ||
                                      strs[0] == 'z' || strs[0] == 's')))
    ok = 1 ;
  return (ok) ;
}

/* In chinese, beside "a", "e", "i", "o", "u", we have two others single vowels
   noted here "yv" and "er" */
/*    check if the character is vowel           */
char Vowel[6] = {'a', 'e', 'i', 'o', 'u', 'y'} ;
int IsVowel (char c)
{
  int ok, i ;

  ok = 0 ;
  for (i=0; i<6; i++)
    if (c == Vowel[i]) {
      ok = 1 ;
      break ;
    }

  return (ok) ;
}
```

```
#include <string.h>
#include <ctype.h>

typedef struct _ChList {
  char phoneme[8] ;
  struct _ChList *next ;
} ChphonemeList;

static int IsVowel (char c) ;
static int IsGroupCSZ (char *strs);
static int Ispunct (char c) ;
static int IsRetroflex (char *strs) ;
static void print_error_malloc (char *varname) ;
static void adjustPinYin (char *strs, int lwd, char *strsnew) ;
static void adjustment_y (char *strs1, int lwd, char *strs2) ;
static void adjustment_w (char *strs1, char *strs2) ;
static void adjustment_u (char *strs1, int lwd, char *strs2) ;
static ChphonemeList *Word2Phonemes (char *word, char tone, ChphonemeList *ListPh) ;
void transform_text_to_phonemes (char *text, char *newtext);

void main ()
{
  char Pinyins[1000], Phonemes[1000] ;
  gets (Pinyins) ;

  transform_text_to_phonemes (Pinyins, Phonemes) ;

  printf ("%s\n", Phonemes) ;
}

void transform_text_to_phonemes (char *text, char *newtext)
{
  int len, i, j ;
  char word[10], neword[10], c, tone ;
  ChphonemeList *ListPh = NULL, *firstP = NULL ;

  strcpy (word,"\0") ;
  strcpy (newtext,"\0") ;

  len = strlen (text) ;

  for (i=0; i<len; i++) {
    c = text[i] ;
    if (isupper(c)) c = tolower (c) ;
    if (c == 'u' && text[i+1] == ':') text[i+1] = 'u' ;
    if ( !isalpha(c) && !Ispunct(c) && !isdigit(c) && c != '#' &&
        c != ' ' && c != '~') {
      printf ("error of Pinyin: %s\n", text) ;
      continue ;
    }
    else if (c == '#' || c == '~' || Ispunct(c)) {
      if (ListPh == NULL) {
        ListPh = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
        if (ListPh == NULL) print_error_malloc ("ListPh") ;
        ListPh->next = NULL ;
        firstP = ListPh ;
      }
      else {
        while (ListPh->next != NULL)
          ListPh = ListPh->next ;
        ListPh->next = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
        ListPh = ListPh->next ;
        if (ListPh == NULL) print_error_malloc ("ListPh->next") ;
        ListPh->next = NULL ;
      }
      if (c == '~')
        strcpy (ListPh->phoneme,"P0\0") ;
      else
        strcpy (ListPh->phoneme,"#\0") ;

    }
    else if (c != ' ' && !isdigit(c))
      strncat (word, &c, 1) ;
    else if (isdigit(c)) {
      tone = c ;
      if (tone == '5') tone = '0' ;
      adjustPinYin (word, strlen(word), neword) ;
      ListPh = Word2Phonemes (neword, tone, ListPh) ;
      if (firstP == NULL) firstP = ListPh ;
```

```
      strcpy (word, "\0") ;
    }
  }

  ListPh = firstP ;
  while (ListPh != NULL) {
    strcat (newtext, ListPh->phoneme) ;
    ListPh = ListPh->next ;
  }

  while (firstP != NULL) {
    ListPh = firstP ;
    firstP = firstP->next ;
    free (ListPh) ;
  }

}

static void print_error_malloc (char *varname)
{
  printf ("error when malloc for the variable \"%s\"\n", varname) ;
  exit (-1) ;
}

static char punct[7]={',', '.', '?','!','"',';', ':'} ;
static int Ispunct (char c)
{
  int i ;

  for (i=0; i<7; i++)
    if (c == punct[i])
      return (1) ;

  return (0) ;
}

static ChphonemeList *Word2Phonemes (char *word, char tone,
                                     ChphonemeList *ListPh)
{
  int ok, okold, len, i, j, Nbvowel ;
  ChphonemeList *lastP = ListPh ;

  if (ListPh == NULL) {
    ListPh = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
    if (ListPh == NULL) print_error_malloc ("ListPh") ;
    ListPh->next = NULL ;
    lastP = ListPh ;
  }
  else {
    while (ListPh->next != NULL)
      ListPh = ListPh->next ;
    ListPh->next = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
    if (ListPh->next == NULL) print_error_malloc ("ListPh->next") ;
    ListPh = ListPh->next ;
    ListPh->next = NULL ;
  }

  Nbvowel = 0 ;
  len = strlen (word) ;
  ok = IsVowel (word[0]);

  if (!ok) {
    ListPh->phoneme[0] = word[0] ;
    ListPh->phoneme[1] = '\0' ;

    for (i=1; i<len; i++) {
      ok = IsVowel (word[i]) ;
      if (!ok)
        strncat (ListPh->phoneme, &word[i],1) ;
      else {
        ListPh->next = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
        if (ListPh->next == NULL) print_error_malloc ("ListPh->next") ;
        ListPh = ListPh->next ;
        ListPh->next = NULL ;
        strcpy (ListPh->phoneme,"\0") ;
        for (j=i; j<len; j++)
          strncat (ListPh->phoneme, &word[j],1) ;
        break ;
      }
```

```
    }
  }
  else
    strcpy (ListPh->phoneme,word) ;

  if (!ok) {
    printf ("error of Pinyin : %s\n", word) ;
    exit(-1) ;
  }

  len = strlen (ListPh->phoneme) ;

  if (ListPh->phoneme[len-1] == 'r' && len > 2) {
    ListPh->phoneme[len-2] = '\0' ;
    if (!strcmp(ListPh->phoneme, "i")) {
      if (IsGroupCSZ (word))
        strcpy (ListPh->phoneme, "<c>i") ;
      else if (IsRetroflex (word))
        strcpy ( ListPh->phoneme, "<ch>i") ;
    }
    strncat (ListPh->phoneme, &tone, 1);
    ListPh->next = (ChphonemeList *)malloc(sizeof(ChphonemeList)) ;
    if (ListPh->next == NULL) print_error_malloc ("ListPh->next") ;
    ListPh = ListPh->next ;
    ListPh->next = NULL ;
    strcpy (ListPh->phoneme,"er\0") ;
  }
  else if (!strcmp(ListPh->phoneme, "i")) {
    if (IsGroupCSZ (word))
      strcpy (ListPh->phoneme, "<c>i") ;
    else if (IsRetroflex (word))
      strcpy ( ListPh->phoneme, "<ch>i") ;
  }

  strncat (ListPh->phoneme, &tone, 1);

  return (lastP) ;
]

/* In chinese, beside "a", "e", "i", "o", "u", we have two others single vowels
   noted here "yv" and "er" */
/*    check if the character is vowel                    */
static char Vowel[6] = ['a', 'e', 'i', 'o', 'u', 'y'] ;
static int IsVowel (char c)
[
  int ok, i ;

  ok = 0 ;
  for (i=0; i<6; i++)
    if (c == Vowel[i]) {
      ok = 1 ;
      break ;
    }

  return (ok) ;
]

static int IsGroupCSZ (char *strs)
[
  int ok=0, len ;

  len = strlen (strs) ;
  if (len == 2 && (strs[0] == 'c' || strs[0] == 's' || strs[0] == 'z'))
    ok = 1 ;
  return (ok) ;
]

static int IsRetroflex (char *strs)
[
  int ok=0, len ;

  len = strlen (strs) ;
  if (strs[0] == 'r' || (len == 3 && (strs[0] == 'c' ||
                                      strs[0] == 'z' || strs[0] == 's')))
    ok = 1 ;
  return (ok) ;
]

static void adjustPinYin (char *strs, int lwd, char *strsnew)
```

```
{
  char firstc, lastc, res[10],c_R ;
  int i ;

  c_R = 'r' ;
  firstc = strs[0] ;
  strsnew[0] = strs[0] ;
  strsnew[1] = '\0' ;
  strcpy (res, "\0") ;
  if (lwd >= 3) {
    for (i=1; i<lwd-2; i++)
      strncat (strsnew, &strs[i],1) ;
    for (i=2; i>0; i--)
      strncat(res,&strs[lwd-i], 1) ;
  }

  if (!strcmp (res, "iu"))
    strcat (strsnew, "iou") ;
  else if (!strcmp(res,"ui"))
    strcat (strsnew, "uei") ;
  else if (!strcmp (res,"un") && firstc != 'y' && firstc != 'x' && firstc != 'q' && firstc != 'j')
    strcat (strsnew, "uen") ;
  else if (!strcmp (res,"ue") && firstc != 'y' && firstc != 'x' && firstc != 'q' && firstc != 'j')
    strcat (strsnew, "yve") ;
  else if (!strcmp(res, "uu"))
    strcat (strsnew, "yv") ;
  else
    strcpy (strsnew, strs) ;

  lwd = strlen (strsnew) ;
  strcpy (res, strsnew) ;

  if (res[lwd-1] == 'r' && lwd > 2) {
    if ((res[lwd-2] != 'e') || IsVowel(res[lwd-3])){
      res[lwd-1] = 'e' ;
      strncat (res, &c_R, 1) ;
    ]
  }

  switch (firstc) {
  case 'y' : adjustment_y (res, lwd, strsnew) ;
    break ;
  case 'w' : adjustment_w (res, strsnew) ;
    break ;
  case 'j' : adjustment_u (res, lwd, strsnew) ;
    break ;
  case 'q' : adjustment_u (res, lwd, strsnew) ;
    break ;
  case 'x' : adjustment_u (res, lwd, strsnew) ;
    break ;
  default : strcpy (strsnew, res) ;
    break ;
  ]



]

/* when the first character of Pinyin is "y", we do the following adjustment */
static void adjustment_y (char *strs1, int lwd, char *strs2)
{
  if (strs1[1] == 'i') {/* here we remove "y" which is not pronounced */
    strcpy(strs2, strchr (strs1, 'i')) ;
  }
  else if (strs1[1] == 'u') {/* here we remove "y" and replace "u" by "yv" */
    strcpy (strs2, "yv");
    if (lwd > 2) strcat (strs2, (strchr (strs1,'u')+1)) ;
  }
  else {/* here "y" is pronounced like "i", we replace it */
    strcpy (strs2, strs1) ;
    strs2[0] = 'i' ;
  ]

}

/* when the first character of Pinyin is "w", we do the following adjustment */
static void adjustment_w (char *strs1, char *strs2)
{
  if (strs1[1] == 'u') /* here we remove "w" which is not pronounced */
```

```
    strcpy (strs2,strchr (strs1, 'u')) ;
  else { /* "w" is pronounced like "u", we repace it */
    strcpy (strs2, strs1) ;
    strs2[0] = 'u' ;
  }

}

/* when the first character of Pinyin is "x" or "q" or "j" and the second
   is "u",  we do the following adjustment */
static void adjustment_u (char *strs1, int lwd, char *strs2)
{
  if (strs1[1] == 'u') { /* here "u" is pronounced like "yv", we repace it */
    strs2[0] = strs1[0] ;
    strs2[1] = '\0' ;
    strcat (strs2, "yv") ;
    if (lwd > 2) strcat (strs2, (strchr (strs1,'u')+1)) ;

  }
  else
    strcpy (strs2, strs1) ;

}
```

```
/*************************************************************************
*                                                                        *
* gblex - translates from GB code to tone pinyin (chinese)        *
*                                                                        *
*          Created in october 1997 by Ming-Yue Xie-Zhang                 *
*************************************************************************/

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

static char c_bdfh[9] ={',', '.',':','"','"','?','!',',',';'};

#define DICT "/home/as62/ming/ch_softw/lex/ma_lex.v03"
#define BDFH "/home/as62/ming/ch_softw/gb2Py/bdfh.gb"
#define MULPIN "/home/as62/ming/ch_softw/gb2Py/mulpin.gb"
#define WORDSIZE 20
#define PHRASESIZE 50

typedef struct _grpInfo {
  int len ;
  char type[30] ;
  char GB[WORDSIZE][2] ;
  char pinyin[WORDSIZE][8] ;
  char tone[WORDSIZE] ;
  struct _grpInfo *next ;
  struct _grpInfo *prev ;
} GrpInfoList;

typedef struct _lexlist {
  char GB[2] ;
  char pinyin[8] ;
  int mulp ;
  GrpInfoList *wdsGrp ;
  struct _lexlist *next ;
} LexiconList ;

typedef struct _mulplist {
  char GB[2] ;
  char pinyin[10][8] ;
  int Nbpy ;
  struct _mulplist *next ;
  struct _mulplist *prev ;
} MulpList ;

LexiconList *firstLT=NULL ;
MulpList *mulpWd ;

LexiconList *read_mandarin_lex (char *name) ;
MulpList *read_mulp_dic (char *name) ;
void reorder_wordsgroup (LexiconList *LexTab) ;
int find_in_mulpin (char *hz, char *pinyin) ;
int IsPunctuation (char c) ;
void print_phrase (char phrase[PHRASESIZE][2], int nn) ;
int translate_phrase_pinyin (char phrase[PHRASESIZE][2],
                             char pinyin[PHRASESIZE][8],
                             int nn);
int search_words (GrpInfoList *current,
                  char phrase[PHRASESIZE][2],
                  char pinyin[PHRASESIZE][8],
                  int i0,
                  int len,
                  int *pnwd0) ;

void main(int argc, char **argv)
{
  register int eka=0, toka=0, i=0, j=0, nn=0;
  register char hz[4] ;
  char bdfh[10][2], namein[80], nameout[80] ;
  char phrase[PHRASESIZE][2], pinyin[PHRASESIZE][8] ;
  char rivi[82];
  int ok_newline, ok_phrase ;
  unsigned char code[82] ;
  register int rpit=0, tila=0 ;
  register LexiconList *LexTab ;
  register GrpInfoList *current, *tmp ;
  FILE *pbdfh = 0, *pfin=0, *pfo=0 ;
```

```
 if (argc < 3) {
  printf ("Usage: %s  name.gb name.pinyin \n", argv[0]) ;
  exit (-1) ;
}
strcpy (namein, argv[1]) ;
strcpy (nameout, argv[2]) ;

if ((pbdfh = fopen(BDFH,"r")) == NULL) {
  fprintf (stderr, "Error: unable to open the punctuation's code file %s\n",
          BDFH);
  exit (-1);
}

for (i=0; i<10; i++) {
  if (!fgets (rivi, 10, pbdfh))
    break ;
  j = 0 ;
  eka = (unsigned char)rivi[j] ;
  while (!(eka & 128)) {
    j++;
    eka = (unsigned char)rivi[j] ;
  }
  bdfh[i][0] = rivi[j] ;
  bdfh[i][1] = rivi[j+1] ;
}
fclose (pbdfh) ;

/* reading the mulpin dict */
mulpWd = read_mulp_dic (MULPIN) ;

/* reading the chinese lexicon */
firstLT = read_mandarin_lex (DICT) ;

/* in each word group, the words are reordered by their lenght */
LexTab = firstLT ;
while (LexTab != NULL) {
  reorder_wordsgroup (LexTab) ;
  LexTab = LexTab ->next ;
}
LexTab = firstLT ;

/* reading the GB text to translate */
pfin = fopen (namein, "r") ;
if (pfin == NULL) {
  printf ("cannot open file %s \n", namein) ;
  exit (-1) ;
}
pfo = fopen (nameout, "w") ;
if (pfo == NULL) {
  printf ("cannot open file %s \n", nameout) ;
  exit (-1) ;
}

hz[2] = '\0';
nn = 0 ;
ok_newline = 0 ;
ok_phrase = 0 ;
while (!feof (pfin)) {
  eka = getc(pfin);
  if (eka < 0) break;

  /* searching for the punctuation symbols or double \newline symbol */
  /* to divide the text in phrases */
  if ((char)eka == '\n') {
    if (ok_newline) {
      ok_phrase = 1 ;
      if (nn > 0) {
        translate_phrase_pinyin (phrase, pinyin, nn) ;
        /*print_phrase (phrase, nn) ;*/
        for (i=0; i<nn; i++)
          fprintf (pfo, "%s ", pinyin[i]) ;
        fprintf (pfo, "\n\n") ;
        nn = 0 ;
      }
    }
    else {
      ok_newline = 1 ;
      if (nn == 0) fprintf (pfo, "\n") ;
```

```
      }
    }
    else if (eka & 128) {
      ok_newline = 0 ;
      toka = getc(pfin);
      if (toka < 0) break;
      if (!(toka & 128)) {
        printf ("input text is not correct GB codage \n").;
        exit (-2) ;
      }
      hz[0] = (char)eka;
      hz[1] = (char)toka;
      tila = -1 ;
      for (i=0; i<9; i++) {
        if (hz[0] != bdfh[i][0] || hz[1] != bdfh[i][1])
          continue;
        else {
          tila=i;
          break ;
        }
      }
      if (tila >= 0) {
        if (nn > 0) {
          translate_phrase_pinyin (phrase, pinyin, nn) ;
          /*print_phrase (phrase, nn) ;*/
          for (i=0; i<nn; i++)
            fprintf (pfo, "%s ", pinyin[i]) ;
          nn = 0 ;
        }
        if (c_bdfh[tila] == '"')
          fprintf (pfo, "\\%c ", c_bdfh[tila]) ;
        else
          fprintf (pfo, "%c ", c_bdfh[tila]) ;
        /*printf ("%c%c",bdfh[tila][0],bdfh[tila][1]) ;*/
      }
      else if ((hz[0] != hz[1]) || (eka != 160 && eka != 161)) {
        phrase[nn][0] = hz[0] ;
        phrase[nn][1] = hz[1] ;
        nn++ ;
      }
    }
    else {
      ok_newline = 0 ;
      if ((char)eka != ' ') {
        if (nn > 0) {
          translate_phrase_pinyin (phrase, pinyin, nn) ;
          /*print_phrase (phrase, nn) ;*/
          for (i=0; i<nn; i++)
            fprintf (pfo, "%s ", pinyin[i]) ;
          nn = 0 ;
        }
        /*printf ("%c", eka) ;*/
        if ((char)eka == '"') fprintf (pfo,"\\") ;
        fprintf (pfo, "%c", eka);
      }
      else if (nn == 0)
        fprintf (pfo," ") ;
    }
  }
  fclose (pfin) ;
  fclose (pfo) ;

  /* free the lists of database */
  LexTab= firstLT ;
  while (LexTab != NULL) {
    current = LexTab->wdsGrp ;
    while (current->prev != NULL) {
      current = current->prev ;
    }
    while (current != NULL) {
      tmp = current ;
      current = current->next ;
      free (tmp) ;
    }
    firstLT = LexTab
    LexTab = LexTab->next ;
    free (firstLT) ;
  }
```

```
}

static LexiconList *tabgrp[256][256] ;

LexiconList *read_mandarin_lex (char *name)
{
  register int i, j ;
  int eka, toka, rpit, tila, nn ;
  int ok_new, ok_err, nbline ;
  char rivi[200], c, type[30], GBcode[2] ;
  unsigned char code[200], ok_first ;
  char strs[10] ;
  register LexiconList *LexTab ;
  register GrpInfoList *current, *tmp ;
  FILE *fopen (), *piffi ;

  if ((piffi = fopen (name, "r")) == 0) {
    fprintf (stderr, "Error: unable to open GB codage dictionnary %s\n",
             name);
    exit (-1);
  ]

  for (i=0; i<256; i++)
    for (j=0; j<256; j++)
      tabgrp[i][j] = NULL ;

  fprintf (stderr, "reading the lexicon........") ;
  nbline = 0 ;
  while (1) {
    if (feof(piffi) || !fgets (rivi, 200, piffi)) [
      fclose (piffi);
      break;
    ]
    nbline++ ;
    rpit = strlen (rivi);
    rivi[rpit-1] = '\0';
    for (i=0; i<rpit; i++)
      code[i] = (unsigned char)rivi[i] ;
    if (rivi[0] == '#') continue; /* for the comment lines */
    ok_err = 0 ;
    eka = code[0];
    i = 0 ;
    if (eka < 0) break;
    while (!(eka & 128)) {
      i++ ;
      eka = code[i] ;
    }
    toka = code[i+1] ;
    if (!(toka & 128)) {
      fprintf (stderr, "DICT \"%s\" :GB code error at line %d \n", nbline) ;
      ok_err = 1 ;
    }
    if (ok_err) continue ;

    GBcode[0] = rivi[i] ;
    GBcode[1] = rivi[i+1] ;
    i += 2 ;
    if (firstLT == NULL) {
      /* allocte the memory for saving the first word groupe */
      if(!(firstLT = (LexiconList *)malloc(sizeof(LexiconList)))) [
        fprintf (stderr, "Error: out of memory when allocating firstLT\n");
        exit (-2);
      }
      LexTab = firstLT ;
      tabgrp[eka][toka] = LexTab ;
      LexTab->mulp = 0 ;
      LexTab->wdsGrp = NULL ;
      LexTab->GB[0] = GBcode[0] ;
      LexTab->GB[1] = GBcode[1] ;
      ok_first = 1 ;
      LexTab->next = NULL ;
    }
    else {
      if (tabgrp[eka][toka] == NULL) {
        /* allocte the memory for saving the new word groupe */
        if (!(LexTab->next =  (LexiconList *)malloc(sizeof(LexiconList)))) {
          fprintf (stderr, "Error: out of memory when allocating LexTab->next\n");
          exit (-2);
        }
```

```
        LexTab = LexTab->next ;
        tabgrp[eka][toka] = LexTab ;
        LexTab->mulp = 0 ;
        LexTab->GB[0] = GBcode[0] ;
        LexTab->GB[1] = GBcode[1] ;
        ok_first = 1 ;
        LexTab->wdsGrp = NULL ;
        LexTab->next = NULL ;
      }
      else {
        LexTab = tabgrp[eka][toka] ;
        ok_first = 0 ;
      }
    }

    if (LexTab->wdsGrp == NULL) {
      /* allocte the memory for saving the first line of the same groupe */
      if(!(LexTab->wdsGrp = (GrpInfoList *)malloc(sizeof(GrpInfoList)))) {
        fprintf (stderr, "Error: out of memory when allocating LexTab->wdsGrp\n");
        exit (-2);
      }
      LexTab->wdsGrp->next = NULL ;
      LexTab->wdsGrp->prev = NULL ;
    }
    else {
      /* allocte the memory for saving the new line of the same groupe */
      if(!(LexTab->wdsGrp->next = (GrpInfoList *)malloc(sizeof(GrpInfoList)))) {
        fprintf (stderr, "Error: out of memory when allocating LexTab->wdsGrp->next\n");
        exit (-2) ;
      }
      current = LexTab->wdsGrp->next ;
      current->prev = LexTab->wdsGrp ;
      LexTab->wdsGrp = LexTab->wdsGrp->next ;
      LexTab->wdsGrp->next = NULL ;
    }

    current = LexTab->wdsGrp ;
    strncpy (current->GB[0], GBcode, 2) ;
    nn = 1 ;

    /* Each line of the lexicon contains the following seven fields.
       Fields are tab separated */
    /* reading now the data field corresponding to the GBcode */
    while (rivi[i] != '\t') {
      eka = code[i] ;
      if (eka & 128) {
        toka = code[i+1] ;
        if (!(toka & 128)) {
          fprintf (stderr, "DICT :GB code error at line %d \n", nbline) ;
          exit(-2) ;
        }
        current->GB[nn][0] = rivi[i] ;
        current->GB[nn][1] = rivi[i+1] ;
        i += 2 ;
        nn++ ;
        if (nn >= WORDSIZE) {
          printf ("nn = %d at line %d: WORDSIZE is not enough\n", nn, nbline) ;
          exit (-1) ;
        }
      }
      else
        i++ ;
    }
    i++ ;
    current->len = nn ;

    /* reading the field of pinyins */
    strcpy (strs,"\0") ;
    nn = 0 ;
    while ((c=rivi[i]) != '\t') {
      if (isalpha (c) || isdigit(c))
        strncat (strs, &c , 1) ;
      else if (c == ' ' && strlen (strs) >= 2) {
        strcpy (current->pinyin[nn], strs) ;
        nn++ ;
        strcpy (strs, "\0") ;
      }
      i++ ;
    }
```

```
    i++ ;
    if (strlen(strs) != 0) {
      strcpy (current->pinyin[nn], strs) ;
      nn++ ;
    }
    if (nn != current->len) {
      fprintf (stderr, "nn= %d DICT :number of PY error at line %d \n", nn, nbline) ;
      exit (-2) ;
    }
    if (ok_first)
      strcpy (LexTab->pinyin, current->pinyin[0]) ;

    nn = 0 ;
    /* reading the tone filed */
    while ((c = rivi[i]) != '\t') {
      if (isdigit (c)) {
        current->tone[nn] = c ;
        nn++ ;
      }
      i++ ;
    }
    i++ ;
    if (nn != current->len) {
      fprintf (stderr, "DICT :number of TONE error at line %d \n", nbline) ;
      exit (-2) ;
    }

    /* go throught the field for the english prounciation  */
    while (rivi[i] != '\t') i++ ;
    i++ ;

    /* go throught the field for the number of occurrences in Xinhua newswire */
    while (rivi[i] != '\t') i++ ;
    i++ ;
    /* go throught the field for the number of occurrences in training transcripts */
    while (rivi[i] != '\t') i++ ;
    i++ ;

    strcpy (type, "\0") ;
    while ((c=rivi[i]) != '\n' && c != '\t') {
      if (c != ' ') strncat (type, &c, 1) ;
      i++ ;
    }
    strcpy (current->type, type) ;

    if (nn == 1)
      LexTab->mulp++ ;
  }/* end of lecture of the DICT */
  fclose (piffi) ;
  printf ("end \n") ;

  return (firstLT) ;
]

int translate_phrase_pinyin (char phrase[PHRASESIZE][2],
                             char pinyin[PHRASESIZE][8],
                             int nn)
{
  register int i, j ;
  register GrpInfoList *current ;
  unsigned char hz[2] ;
  register LexiconList *LexTab ;
  char strs[10] ;
  int nwd0, nwd, match_len, ngrp ;

  for (i=0; i<nn; i++)
    strcpy (pinyin[i],"\0") ;

  strcpy (strs,"\0") ;
  i = 0 ;
  match_len = nn  ;
  while( i<nn) {
    nwd0 = 0 ;
    hz[0] = (unsigned char)phrase[i][0] ;
    hz[1] = (unsigned char)phrase[i][1] ;
    LexTab = tabgrp[hz[0]][hz[1]] ;
    if (!LexTab) [
      strcpy (pinyin[i], " ") ;
      i++ ;
```

```
      continue ;
    }
    if (match_len == 1) {
      strcpy (pinyin[i], LexTab->pinyin) ;
      return (1) ;
    }
    current = LexTab->wdsGrp ;
    while (current->prev != NULL)
      current = current->prev ; /* go to the first line of this groupe */

    while (current != NULL) {
      nwd = search_words (current, phrase, pinyin, i, match_len, &nwd0);
      if (nwd == current->len)
        break ;
      if (nwd0 <= match_len) {
        if (current->next == NULL) break ;
        current = current->next ;
        if (current->len < nwd0) break ;
      }
      else
        break ;
    }

    if (nwd0 == 1) {
      strcpy (pinyin[i], LexTab->pinyin) ;
    }
    i += nwd0 ;
    match_len = nn - i ;
  }

  return (1) ;
}

int search_words (GrpInfoList *current,
                  char phrase[PHRASESIZE][2],
                  char pinyin[PHRASESIZE][8],
                  int i0,
                  int len,
                  int *pnwd0)
{
  int i, nwd, ii, l ;

  nwd = 0 ;
  if (current->len < len) len = current->len ;
  for (i=0; i<len; i++) {
    if (current->GB[i][0] != phrase[i0+i][0] ||
        current->GB[i][1] != phrase[i0+i][1])
      break ;
    nwd++ ;
  }

  if (nwd >= *pnwd0) {
    for (i=0; i<nwd; i++){
      ii = i + i0 ;
      strcpy(pinyin[ii], current->pinyin[i]) ;
      if (nwd == current->len) {
        l = strlen(pinyin[ii]) ;
        pinyin[ii][l-1] = current->tone[i] ;
      }
    }
    *pnwd0 = nwd ;
    return (nwd) ;
  }

  return (nwd) ;

}

void reorder_wordsgroup (LexiconList *LexTab)
{
  register GrpInfoList *current, *tmp ;
  int i, len0, maxlen, ok, tila ;
  char tone[WORDSIZE], pinyin[WORDSIZE][8], code[WORDSIZE][2] ;
  char strs[8] ;

  if (LexTab->mulp > 1) {
    tila = find_in_mulpin (LexTab->GB, strs) ;
    if (tila)  {
      strcpy (LexTab->pinyin, strs) ;
```

```
    ]
  }

  current = LexTab->wdsGrp ;
  while (current->prev != NULL)
    current = current->prev ;
  LexTab->wdsGrp = current ;

  while (1) {
    current = LexTab->wdsGrp ;
    if (current->next == NULL)
      break ;
    len0 = current->len ;
    maxlen = len0 ;
    ok = 0 ;
    while (current->next != NULL) {
      current = current->next ;
      if (current->len > maxlen) {
        maxlen = current->len ;
        tmp = current ;
        ok = 1 ;
      }
    }
    current = LexTab->wdsGrp ;
    if (ok) {
      current->len = maxlen ;
      for (i=0; i<len0; i++) {
        strcpy (pinyin[i], current->pinyin[i]) ;
        code[i][0] = current->GB[i][0] ;
        code[i][1] = current->GB[i][1] ;
        tone[i} = current->tone[i] ;
      }
      for (i=0; i<maxlen; i++) {
        current->GB[i][0] = tmp->GB[i][0] ;
        current->GB[i][1] = tmp->GB[i][1] ;
        strcpy (current->pinyin[i], tmp->pinyin[i]) ;
        current->tone[i] = tmp->tone[i] ;
      }
      tmp->len = len0 ;
      for (i=0; i<len0; i++) {
        tmp->GB[i][0] = code[i][0] ;
        tmp->GB[i][1] = code[i][1] ;
        strcpy (tmp->pinyin[i], pinyin[i]) ;
        tmp->tone[i] = tone[i] ;
      ]
    }
    if (LexTab->wdsGrp->next == NULL) break ;
    LexTab->wdsGrp = LexTab->wdsGrp->next ;
  }

  current = LexTab->wdsGrp ;
  while (current->prev != NULL)
    current = current->prev ;
  LexTab->wdsGrp = current ;

}

/* looking for the character in the lexicon of characters with multiple pronunciations */
int find_in_mulpin (char *hz, char *pinyin)
{
  int tila ;

  tila = 0 ;
  while (mulpWd->prev != NULL)
    mulpWd = mulpWd->prev ;

  if (mulpWd == NULL) {
    printf ("error of mulpy\n") ;
    exit (-1) ;
  }

  while (mulpWd != NULL) {
    if (hz[0] == mulpWd->GB[0] && hz[1] == mulpWd->GB[1]) {
      strcpy (pinyin, mulpWd->pinyin[0]) ;
      tila = 1 ;
      break ;
    }
    else {
      if (mulpWd->next == NULL) break ;
```

```
      mulpWd = mulpWd->next ;
    }
  ]

  return (tila) ;
]

/* reading the lexicon which give the more frequent pronunciation for the characters which have mult
iple pronunciations */
MulpList *read_mulp_dic (char *name)
{
  char line[250] ;
  int eka, toka, i, npy ;
  char hz[4], c ;
  MulpList *firstWd = NULL, *mulpWd = NULL ;
  FILE *pmulp ;

  if ((pmulp = fopen(name,"r")) == NULL) {
    fprintf (stderr, "Error: unable to open multi-pinyin dictionnary %s\n",
            name);
    exit (-1);
  ]

  hz[2] = '\0' ;

  do [
    eka = getc(pmulp) ;
    if (eka < 0) {
      printf ("irregular character \n") ;
      exit (-1) ;
    ]
  }while (eka != EOF && !(eka & 128)) ;

  while ((toka =getc(pmulp)) != EOF) [
    if (toka < 0) {
      printf ("irregular character \n") ;
      exit (-1) ;
    ]
    else if (toka & 128) {
      hz[0] = (char)eka ;
      hz[1] = (char)toka ;
      if (firstWd == NULL) {
        mulpWd = (MulpList *)malloc(sizeof(MulpList)) ;
        mulpWd->prev = NULL ;
        mulpWd->next = NULL ;
        firstWd = mulpWd ;
      ]
      else [
        mulpWd->next = (MulpList *)malloc(sizeof(MulpList)) ;
        mulpWd->next->prev = mulpWd ;
        mulpWd = mulpWd->next ;
        mulpWd->next = NULL ;
      ]
      mulpWd->GB[0] = hz[0] ;
      mulpWd->GB[1] = hz[1] ;
      npy = 0 ;
      strcpy (mulpWd->pinyin[0], "\0") ;
      while ((eka = getc(pmulp)) != EOF) [
        if (eka & 128)
          break ;
        c = (char)eka ;
        if (npy >= 10) [
          printf ("Error: out of memory npy= %d\n", npy) ;
          for (i=0; i<10; i++)
            printf ("%s ", mulpWd->pinyin[i]) ;
          printf ("\n") ;
          exit (-1) ;
        ]
        if (isalpha(c))
          strncat (mulpWd->pinyin[npy], &c, 1);
        else if (isdigit(c)) [
          strncat (mulpWd->pinyin[npy], &c, 1);
          npy++ ;
          strcpy (mulpWd->pinyin[npy], "\0") ;
        ]
      ]
      mulpWd->Nbpy = npy ;
    ]
    else [
```

```
      printf ("multi-dic error\n") ;
      exit (-1) ;
    }
  }
  fclose (pmulp) ;

  return (firstWd) ;


}

void print_phrase (char phrase[PHRASESIZE][2], int nn)
{
  register int i ;

  printf ("\n") ;
  for (i=0; i<nn; i++)
    printf ("%c%c", phrase[i][0], phrase[i][1]) ;
}

static char punct[7] = ['.', ',',':',';','!','?','"'];

int IsPunctuation (char c)
{
  int i, ok=0 ;

  for (i=0; i<7; i++)
    if (c == punct[i]) {
      ok = 1 ;
      break ;
    }

  return (ok) ;
}
```