Converting from traditional to simplified Chinese, part 1: Loading the dictionary

devblogs.microsoft.com/oldnewthing/20050711-14

July 11, 2005

Raymond Chen

One step we had glossed over in our haste to get something interesting on the screen in our Chinese/English dictionary program was the conversion from traditional to simplified Chinese characters.

The format of the `hcutf8.txt` file is a series of lines, each of which is a UTF-8 encoded string consisting of a simplified Chinese character followed by its traditional equivalents. Often, multiple traditional characters map to a single simplified character. Much more rarely—only twice in our data set—multiple simplified characters map to a single traditional character. Unfortunately, one of the cases is the common syllable 麼, which has two simplifications, either 么 or 麽, the first of which is far more productive. We'll have to keep an eye out for that one.

(Note also that in real life, the mapping is more complicated than a character-for-character substitution, but I'm willing to forego that level of complexity because this is just for my personal use and people will have realized I'm not a native speaker long before I get caught up in language subtleties like that.)

One could try to work out a fancy data structure to represent this mapping table compactly, but it turns out that simple is better here: an array of 65536 `WCHAR` s, each producing the corresponding simplification. Most of the array will lie unused, since the characters we are interested in lie in the range U+4E00 to U+9FFF. Consequently, the active part of the table is only about 40Kb, which easily fits inside the L2 cache.

It is important to know when a simple data structure is better than a complex one.

The `hcutf8.txt` file contains a lot of fluff that we aren't interested in. Let's strip that out ahead of time so that we don't waste our time parsing it at run-time.

```
#!perl
$_ = <> until /^# Start zi/; # ignore uninteresting characters
while (<>) {
 s/\r//g;
 next if length($_) == 7 &&
         substr($_, 0, 3) eq substr($_, 3, 3); # ignore NOPs
 print;
}
```

Run the `hcutf8.txt` file through this filter to clean it up a bit.

Now we can write our “traditional to simplified” dictionary.

```
class Trad2Simp
{
public:
 Trad2Simp();
 WCHAR Map(WCHAR chTrad) const { return _rgwch[chTrad]; }


private:
 WCHAR _rgwch[65536]; // woohoo!
};


Trad2Simp::Trad2Simp()
{
 ZeroMemory(_rgwch, sizeof(_rgwch));


 MappedTextFile mtf(TEXT(“hcutf8.txt”));
 const CHAR* pchBuf = mtf.Buffer();
 const CHAR* pchEnd = pchBuf + mtf.Length();
 while (pchBuf < pchEnd) {
  const CHAR* pchCR = std::find(pchBuf, pchEnd, ‘\r’);
  int cchBuf = (int)(pchCR – pchBuf);
  WCHAR szMap[80];
  DWORD cch = MultiByteToWideChar(CP_UTF8, 0, pchBuf, cchBuf,
                                  szMap, 80);
  if (cch > 1) {
   WCHAR chSimp = szMap[0];
   for (DWORD i = 1; i < cch; i++) {
    if (szMap[i] != chSimp) {
     _rgwch[szMap[i]] = chSimp;
    }
   }
   pchBuf = std::find(pchCR, pchEnd, ‘\n’) + 1;
  }
 }
 _rgwch[0x9EBC] = 0x4E48;
}
```

We read the file one line at a time, convert it from UTF-8, and for each nontrivial mapping, record it in our dictionary. At the end, we do our little 么 special-case patch-up.

Next time, we’ll use this mapping table to generate simplified Chinese characters into our dictionary.

Raymond Chen

Follow