

## UTF-8 installations of CWEB

Igor Liferenko

### Abstract

We show how to implement UTF-8 support in CWEB [1] by adding the arrays *xord* and *xchr*. Immediately after reading a Unicode character from an input file, we convert it to an 8-bit character using *xord*. On output the reverse operation is done using *xchr*. This allows us to leave core algorithms of CWEB unchanged.

Incidentally, the described method allows to use the extended character set [1] of CWEB: the characters ‘↑’, ‘↓’, ‘→’, ‘←’, ‘≠’, ‘≤’, ‘≥’, ‘≡’, ‘√’, ‘∧’, ‘C’, and ‘D’ can be typed as abbreviations for C language digraphs ‘++’, ‘--’, ‘->’, ‘!=’, ‘<=’, ‘>=’, ‘==’, ‘||’, ‘&&’, ‘<<’, and ‘>>’, respectively.

### 1. Initialization

(For brevity, in the diffs following, the original code in the CWEB source is preceded with < characters, and the new code with >. Both are sometimes reformatted for presentation in this article, and for readability we sometimes leave a blank line between the pieces. The actual implementation uses the change files `comm-utf8.ch`, `cweav-utf8.ch` and `ctang-utf8.ch`, together with `common-utf8.ch` [2].)

First, we add global arrays *xord* and *xchr* to `common.w` [1]. We declare the size of the *xord* array to be  $2^{16}$  bytes. This means that only values from the basic multilingual plane (BMP) of Unicode are permitted. We use the `wchar_t` data type for characters in input files to accommodate Unicode values.

Background: this predefined C type allocates four bytes per character (on most systems). Character constants of this type are written as `L'...'`.

```
unsigned char xord[65536];
wchar_t xchr[256];
```

These same arrays must be used in `cweave.w` [1].

```
extern unsigned char xord[];
extern wchar_t xchr[];
```

In `ctangle.w` [1] only the *xchr* array is needed.

```
extern wchar_t xchr[];
```

We initialize the *xord* and *xchr* arrays in the `common_init` function of `common.w`. First, in *xchr* we map all ASCII characters to themselves:

```
for (i=0; i<=0176; i++) xchr[i]=i;
```

Then we map the rest of the indexes of *xchr* to space.

```
for (i=0177; i<=0377; i++) xchr[i]=' ';
```

Elements in the *xchr* array are overridden using the file `mapping.w` [2].

```
@i mapping.w
```

This file specifies the character(s) required for a particular installation of CWEB, for example:

```
xchr[0xa7] = L'ë';
```

The initialization of *xord* comes next. All its indexes are mapped by default to 127, which is the ASCII character code (DEL) that is prohibited in text files. Then we make *xord* contain the inverse of the information in *xchr*.

```
for (i=0; i<=65535; i++) xord[i]=127;
for (i=0200; i<=0377; i++) xord[xchr[i]]=i;
for (i=0; i<=0176; i++) xord[xchr[i]]=i;
```

It remains to set the `LC_CTYPE` locale category. The behavior of the C library functions used below depends on this value.

```
setlocale(LC_CTYPE, "C.UTF-8");
```

Finally, we need the necessary headers.

```
#include <wchar.h>
#include <locale.h>
```

### 2. Input

For automatic conversion from UTF-8 to Unicode, we change the `input_ln` function to use `fgetwc` [3] instead of `getc`. Also, `ungetc` is changed to `ungetwc` [3] and `EOF` must be replaced with `WEOF` [3] (for this, `int` is changed to `wint_t` [3]).

```
< int c;
> wint_t c;

< while (k<=buffer_end && (c=getc(fp))
<   != EOF && c!='\n')
> while (k<=buffer_end && (c=fgetwc(fp))
>   != WEOF && c!=L'\n')

< if ((c=getc(fp))!=EOF && c!='\n') {
> if ((c=fgetwc(fp))!=WEOF && c!=L'\n') {

< ungetc(c,fp);
> ungetwc(c,fp);

< if (c==EOF && limit==buffer) return(0);
> if (c==WEOF && limit==buffer) return(0);
```

The conversion with *xord* is done immediately after a character is read.

```
< if ((*k++) = c) != ' ') limit = k;
> if ((*k++) = xord[c]) != ' ') limit = k;
```

### 3. Output

We use *xchr* and *printf* with `%lc` conversion specifier for characters, printed on terminal during error reporting.

```
< putchar(*k);
> printf("%lc",xchr[(unsigned char)*k]);
```

The *term\_write* macro uses the C library function *fwrite* to output a range of characters. We must use *xchr* for each character (except the newline character), then convert it to UTF-8 via *printf*, using `%lc` conversion specifier.

```
< @d term_write(a,b) fflush(stdout),
<     fwrite(a,sizeof(char),b,stdout)

> @d term_write(a,b) do { fflush(stdout);
>   for (int i = 0; i < b; i++)
>     if (*(a+i)=='\n') putchar('\n');
>     else printf("%lc",xchr[(unsigned char)
>       *(a+i)]); } while (0)
```

In *cweave.w* all output to files is done via the *c\_line\_write* macro. This uses the C library function *fwrite* to output a range of characters. Since *xchr* must be used for each character, we loop over this range and convert each character to the external encoding and then to UTF-8 via *fprintf*, using the `%lc` conversion specifier.

```
< fwrite(out_buf+1,sizeof(char),c,
<   active_file)

> for (int i = 0; i < c; i++)
>   fprintf(active_file, "%lc",
>     xchr[(eight_bits) *(out_buf+1+i)])
```

Similarly, in *ctangle.w*, before outputting characters in C string constants, convert each of them to the external encoding and then to UTF-8 using the `%lc` conversion specifier of *fprintf*.

```
< C_putc(a);
> fprintf(C_file,"%lc",xchr[(eight_bits)a]);
```

For other output code no special treatment is needed, since all other output data is in ASCII, which is part of UTF-8 (except file names, which are already in UTF-8).

### 4. The file name buffer

File names must be in UTF-8. So, before appending characters to *cur\_file\_name*, we convert them to the external encoding and then to UTF-8 via C library function *wctomb* [3].

```
< *k++=*loc++;

> { char mb[MB_CUR_MAX]; int len =
```

```
>   wctomb(mb,xchr[(unsigned char)*loc++]);
>   if (k<=cur_file_name_end)
>     for (int i = 0; i<len; i++) *k++=mb[i];
>   else k=cur_file_name_end+1; }
```

### 5. Locale considerations

*cweave.w* uses the locale-dependent C library functions *islower*, *isupper* and *tolower* (the former two via *xislower* and *xisupper* macros respectively). But since we are assuming the UTF-8 locale, instead of these we must use *iswlower*, *iswupper* and *towlower* from *wctype.h* [3]. The trick is to convert from the internal encoding to the external encoding before using these functions.

```
< xislower(*x)
> iswlower(xchr[(eight_bits)*x])

< xisupper(x)
> iswupper(xchr[(eight_bits) x])
```

For *towlower* the result must be converted back from the external encoding to the internal encoding.

```
< c=tolower(c)
> c=xord[towlower(xchr[(eight_bits)c])]
```

### References

- [1] Knuth, D. and Levy, S. The CWEB System of Structured Documentation, 1993. ISBN 0-201-57569-8
- [2] Source of the present implementation. <https://github.com/igor-liferenko/cweb>
- [3] Single Unix Specification. Introduction to ISO C Amendment 1 (Multibyte Support Environment). [https://unix.org/version2/whatsnew/login\\_mse.html](https://unix.org/version2/whatsnew/login_mse.html)

◇ Igor Liferenko  
igor.liferenko (at) gmail dot com