2002/2003 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
A Compression Scheme for Unicode

Character strings on a computer are handled by a mapping between characters and numbers as well as a scheme for representing the sequence of numbers as a stream of bytes on the computer. The Unicode standard mapping (also known as ISO 10646) is designed to provide a universal mapping between characters and numbers for essentially all characters for all writing systems for all history. This mapping has the capability to make all other mappings obsolete, and is in widespread use throughout the world.

There are many different ways of representing a sequence of numbers as a stream of bytes. One scheme, known as UTF-16LE, uses pairs of bytes in little endian order to represent unsigned integers. Unwilling to use two bytes per character, your customer paid a high priced consulting firm to compress their UTF-16LE data.

Falling memory prices and the difficulties of manipulating compressed data has convinced your customer to convert the compressed Unicode data back to UTF-16LE. Your task is to write a program that inputs a sequence of Compressed Unicode strings, and outputs UTF-16LE. The Unicode Line Feed character (mapped to the number 10) indicates the end of a string. Each string was compressed individually. The high priced consulting firm did a competent job, so you can assume all input strings are well formed and well defined.

The compression scheme operates by providing two compression modes: single byte mode and Unicode mode. Sequences of control bytes switch between compression modes and alter the interpretation of subsequent bytes. The compression scheme uses *windows* into the Unicode mapping. There are 16 windows used for interpretation of compressed strings: eight static windows (SW0 - SW7) and eight dynamic windows (DW0 - DW7). Each window includes 128 sequential characters from the Unicode map. Different windows can have different starting positions.

A window is completely determined with respect to Unicode by its starting character. SW0 starts with the character mapped to 0, so the value 65 with respect to SW0 is the character mapped to Unicode at 65. SW1 starts with the character that maps to 128, so the value 65 with respect to SW1 is the character mapped to Unicode at 193. SW2 starts at 256, SW3 at 768, SW4 at 8192, SW5 at 8320, SW6 at 8448, and SW7 at 12288. DW0 *initially* starts at 128, DW1 at 192, DW2 at 1024, DW3 at 1536, DW4 at 2304, DW5 at 12352, DW6 at 12448, and DW7 at 65280. Dynamic windows can be repositioned via control sequences.

Single byte mode has each byte representing a character. The byte values 0, 9, 10, 13, as well as 32 through 127 are interpreted as Unicode (e.g., the byte with value 10 is interpreted as Line Feed). The byte values 128 through 255 are interpreted as the characters numbered 0 through 127 in the current window (initially DW0). All other byte values indicate control bytes explained later. Values 11 and 12 are not used in this problem.

Unicode mode has pairs of bytes in *big* endian form, interpreted as Unicode. For example, the two-byte sequence 0, 10 is interpreted as Line Feed. The first byte of a byte pair is not allowed to have values from 224 through 242. Such bytes are control bytes explained later. Values 241 and 242 are not used for this problem.

Input to the program is a *binary* file that contains Compressed Unicode strings, terminated by end-of-file. When interpreting a compressed string, interpretation begins in single byte mode with DW0 as the current window, and all dynamic windows begin with their *initial* starting positions as described above. Interpretation changes only when a control byte is encountered.

Output is a *binary* file that contains (little endian) UTF-16LE.

Problem 6
A Compression Scheme for Unicode (continued)

The notation that follows includes mnemonics consisting of two uppercase characters followed by a digit. In this context, the use of these same two characters followed by a lower case ‘n’ indicates any of those mnemonics. However, the digit indicated by ‘n’ is fixed for subsequent usage within the same paragraph.

Control bytes for single byte mode:

Mnemonic	*Byte Value(s)*	*Function*
SQ0–SQ7	1–8	Let Vbyte be the value of the byte following the SQn. If $0 \le$ Vbyte ≤ 127, then this represents character Vbyte in SWn. If $128 \le$ Vbyte ≤ 255, then this represents the character Vbyte $-$ 128 in DWn.
SQU	14	The pair of bytes following SQU should be interpreted as big endian Unicode.
SCU	15	Enter Unicode mode.
SC0–SC7	16–23	SCn changes the current window to DWn
SD0–SD7	24–31	The byte following SDn is used to define the starting position of DWn (See window offset table). DWn becomes the current window.

Control bytes for Unicode mode:

Mnemonic	*Byte Value(s)*	*Function*
UC0–UC7	224–231	UCn changes the current window to DWn and enters single byte mode.
UD0–UD7	232–239	The byte following UDn is used to define the starting position of DWn (See window offset table). DWn becomes the current window and single byte mode is entered.
UQU	240	The pair of bytes following UQU should be evaluated as big endian Unicode.

Window offset table:

Byte Value(s)	*Offset*
0	Not to be used
1–103	Byte value multiplied by 128
104–167	Byte value multiplied by 128, added to 44032
168–248	Not to be used
249	192
250	592
251	880
252	1328
253	12352
254	12448
255	65376

The Sample Input and Sample Output are unsigned decimal dumps of the sample binary files. Use the command **`od -A x -t u1 filename`** *to display the contents of a binary file as unsigned decimal byte values.*

Sample Input

```
0000000 065 194 010 001 001 010 014 078 010 078 010 015 078 010 078 010
0000010 000 010 018 001 194 194 010 026 251 001 194 194 018 194 014 078
0000020 010 194 010 194 010
0000025
```

Sample Output

```
0000000 065 000 194 000 010 000 001 000 010 000 010 078 078 000 010 000
0000010 010 078 010 078 010 000 194 000 066 004 010 000 194 000 178 003
0000020 178 003 010 078 178 003 010 000 194 000 010 000
000002c
```