

## View Files

Contents

1. How to start . . . . .	2
2. Screen contents . . . . .	3
3. How to . . . . .	4
3.1. Exit the viewer . . . . .	4
3.2. Scroll the file . . . . .	4
3.3. Toggle line wrapping . . . . .	5
3.4. How to change encoding . . . . .	9
3.5. How to search for a text in the file . . . . .	12

## 1. How to start

To view a file, set cursor over it using ↑ and ↓ keys or by clicking it with left mouse button. Then press the **F3** key.

```

Name      Z:\
_test
Bin
Devices
KernelModules
Locales
Processes
SharedLibraries
System

Name      Z:\_test
..
empty.txt
init2
one.txt
UTF-16-test.txt
UTF-8-demo.html
UTF-8-test.txt

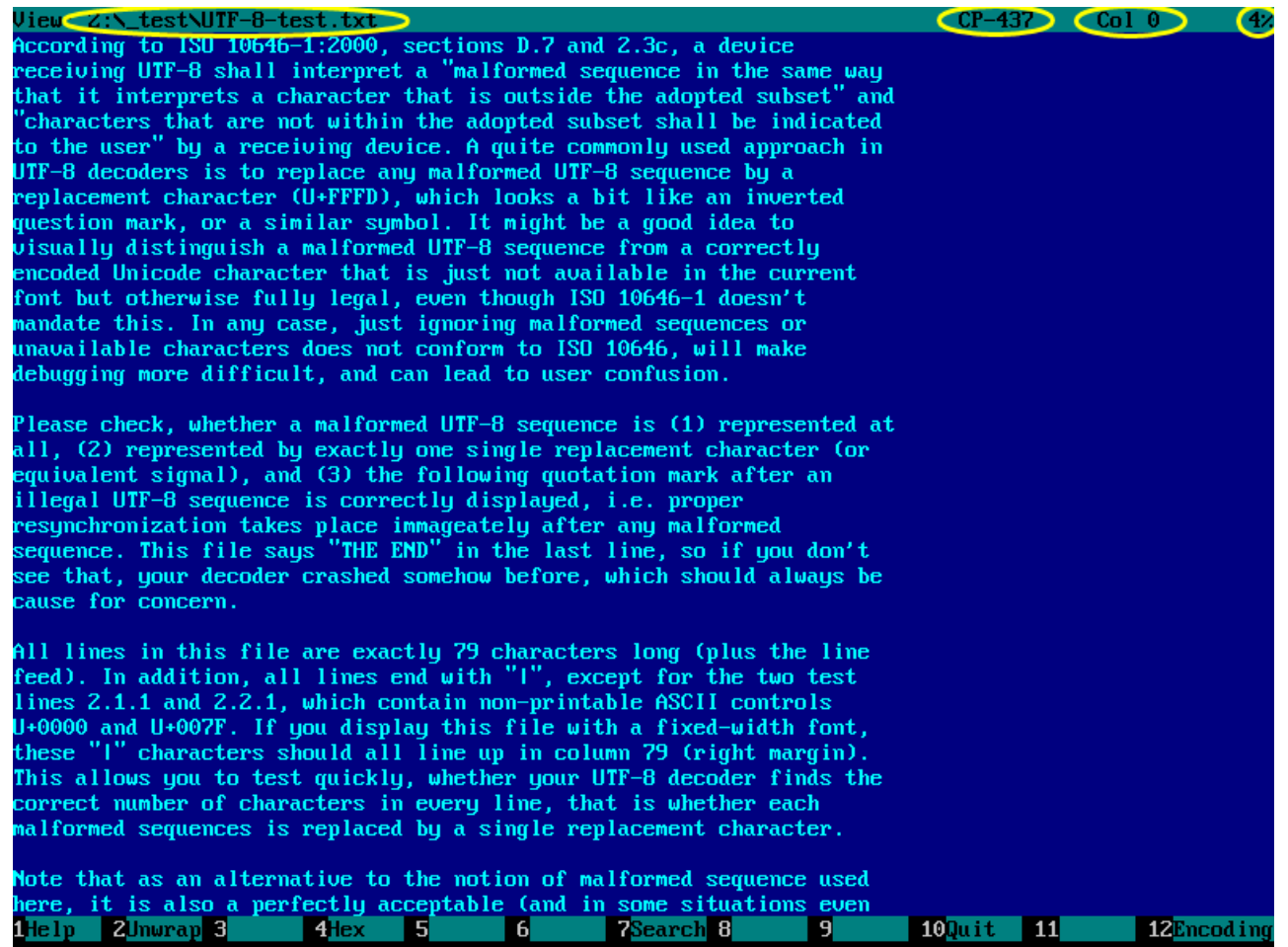
_test      Folder 05/11/08 10:55
UTF-8-test.txt  20334 05/11/08 10:55

1  2  3 View  4 Edit  5 Copy  6 RenMou  7 Create  8 Delete  9  10 Quit  11  12
```

## 2. Screen contents

Topmost line of the viewer contains:

- name of the file being viewed
- encoding
- current column number
- offset of the first visible line, in percent



The screenshot shows a terminal window with a dark blue background and light blue text. At the top, a status bar displays 'View Z:\N\_test\UTF-8-test.txt' (highlighted in yellow), 'CP-437' (highlighted in yellow), 'Col 0' (highlighted in yellow), and '4%' (highlighted in yellow). The main text area contains three paragraphs of text. The first paragraph discusses ISO 10646-1:2000 sections D.7 and 2.3c, explaining how a device receiving UTF-8 should interpret malformed sequences. The second paragraph asks the user to check if a malformed UTF-8 sequence is represented by a single replacement character, a single quotation mark, or correctly displayed. The third paragraph states that all lines in the file are 79 characters long (plus a line feed), except for two test lines (2.1.1 and 2.2.1) which contain non-printable ASCII controls. The bottom of the screen shows a menu with numbered options: 1 Help, 2 Unwrap, 3, 4 Hex, 5, 6, 7 Search, 8, 9, 10 Quit, 11, 12 Encoding.

```
View Z:\N_test\UTF-8-test.txt CP-437 Col 0 4%
According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving device. A quite commonly used approach in
UTF-8 decoders is to replace any malformed UTF-8 sequence by a
replacement character (U+FFFD), which looks a bit like an inverted
question mark, or a similar symbol. It might be a good idea to
visually distinguish a malformed UTF-8 sequence from a correctly
encoded Unicode character that is just not available in the current
font but otherwise fully legal, even though ISO 10646-1 doesn't
mandate this. In any case, just ignoring malformed sequences or
unavailable characters does not conform to ISO 10646, will make
debugging more difficult, and can lead to user confusion.

Please check, whether a malformed UTF-8 sequence is (1) represented at
all, (2) represented by exactly one single replacement character (or
equivalent signal), and (3) the following quotation mark after an
illegal UTF-8 sequence is correctly displayed, i.e. proper
resynchronization takes place immediately after any malformed
sequence. This file says "THE END" in the last line, so if you don't
see that, your decoder crashed somehow before, which should always be
cause for concern.

All lines in this file are exactly 79 characters long (plus the line
feed). In addition, all lines end with "l", except for the two test
lines 2.1.1 and 2.2.1, which contain non-printable ASCII controls
U+0000 and U+007F. If you display this file with a fixed-width font,
these "l" characters should all line up in column 79 (right margin).
This allows you to test quickly, whether your UTF-8 decoder finds the
correct number of characters in every line, that is whether each
malformed sequences is replaced by a single replacement character.

Note that as an alternative to the notion of malformed sequence used
here, it is also a perfectly acceptable (and in some situations even
1 Help 2 Unwrap 3 4 Hex 5 6 7 Search 8 9 10 Quit 11 12 Encoding
```

## 3. How to

### 3.1. Exit the viewer

Press **F10** or **Esc** key to exit the viewer.

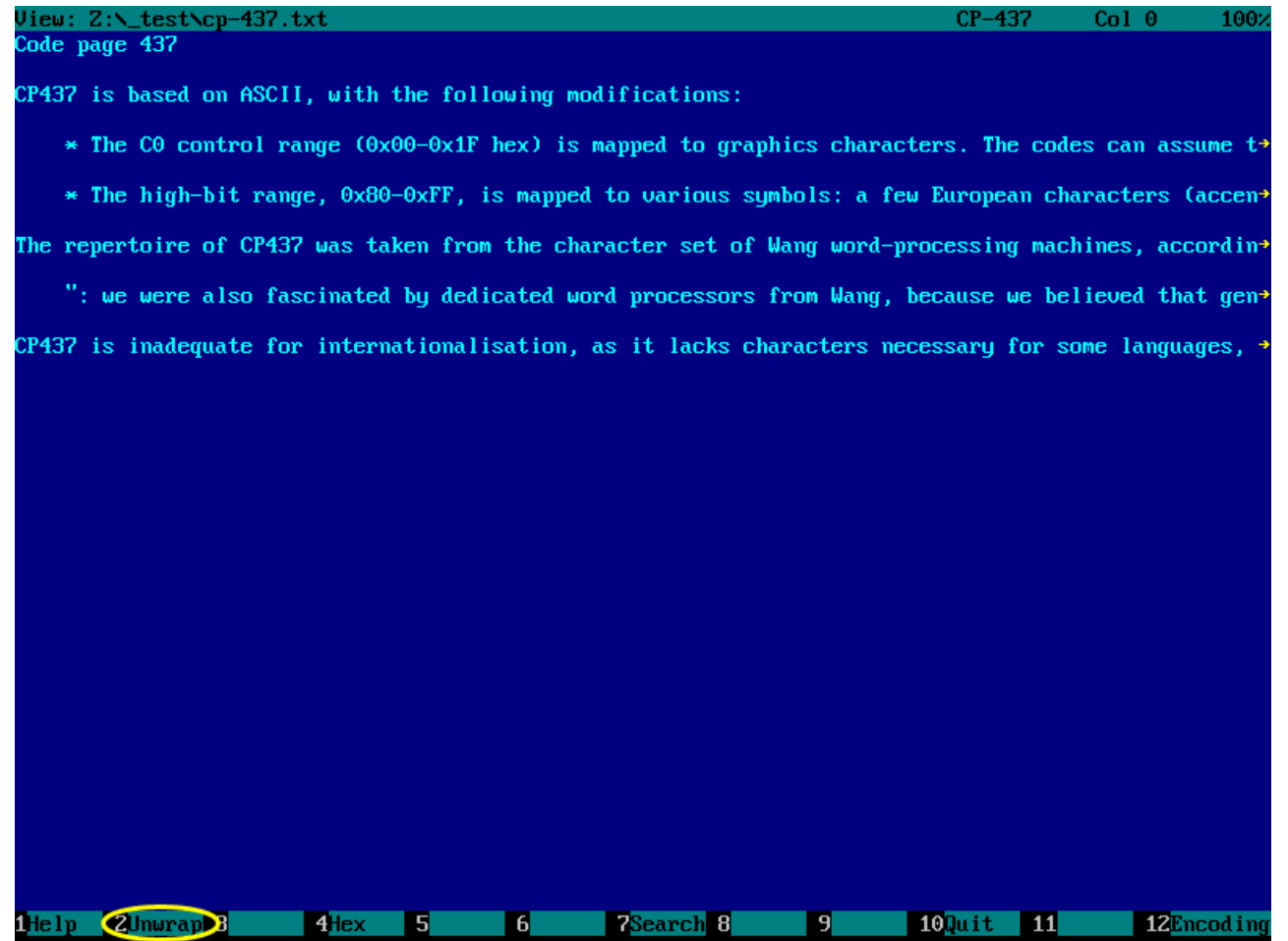
### 3.2. Scroll the file

Use the following keys to scroll the file:

Operation	Key
Scroll up by one line	↑
Scroll down by one line	↓
Scroll left by one column	←
Scroll right by one column	→
Scroll up by one page	<b>PageUp</b>
Scroll down by one page	<b>PageDown</b>
Move to the start of the file	<b>Home</b> or <b>Ctrl+Home</b> or <b>Ctrl+PageUp</b>
Move to the end of the file	<b>End</b> or <b>Ctrl+End</b> or <b>Ctrl+PageDown</b>

### 3.3. Toggle line wrapping

To toggle line wrapping in the viewer, use the **F2** key.



```
View: Z:\_test\cp-437.txt          CP-437    Col 0    100%
Code page 437

CP437 is based on ASCII, with the following modifications:

    * The C0 control range (0x00-0x1F hex) is mapped to graphics characters. The codes can assume t→
    * The high-bit range, 0x80-0xFF, is mapped to various symbols: a few European characters (accen→

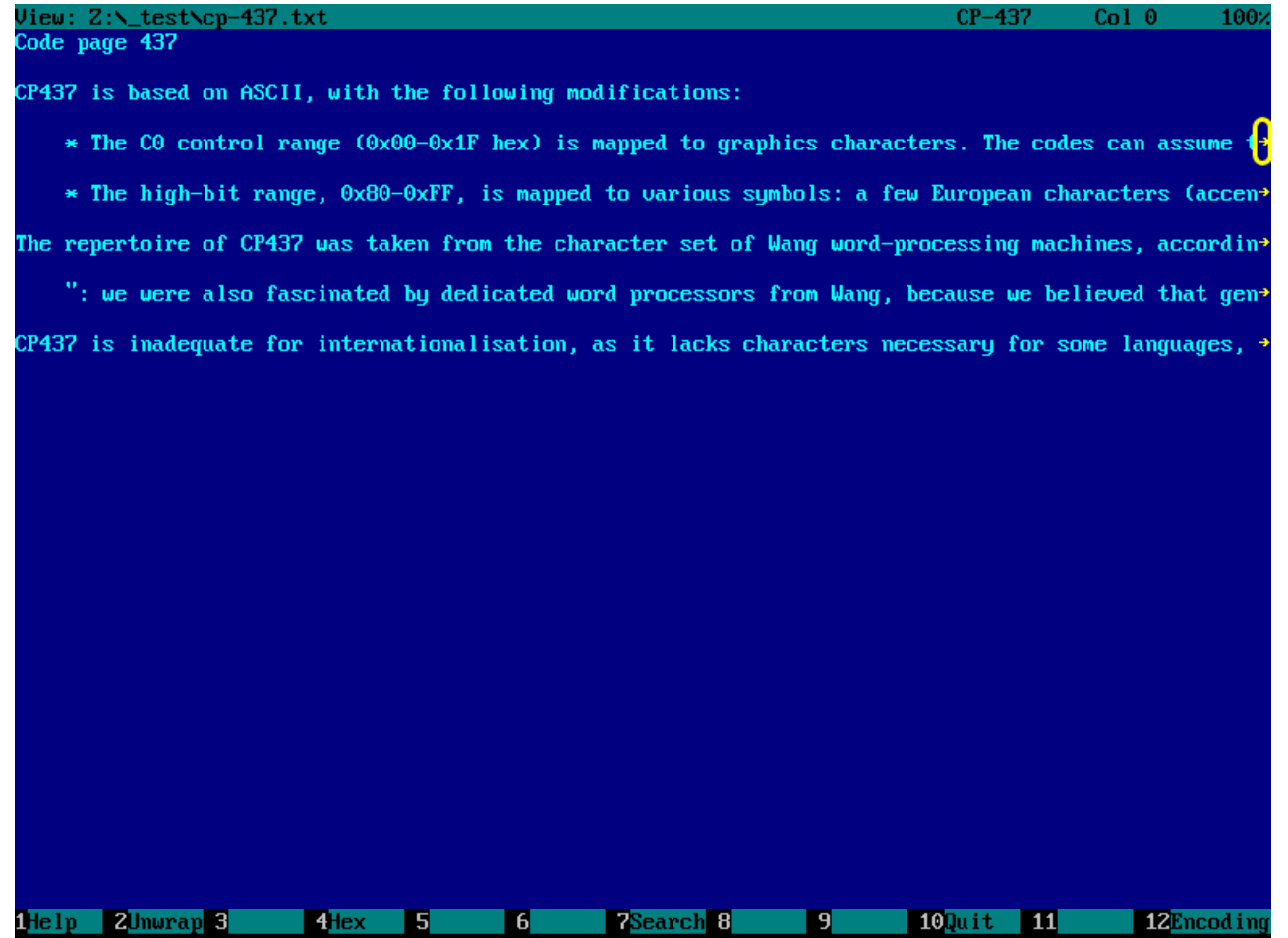
The repertoire of CP437 was taken from the character set of Wang word-processing machines, accordin→

    ": we were also fascinated by dedicated word processors from Wang, because we believed that gen→

CP437 is inadequate for internationalisation, as it lacks characters necessary for some languages, →

1 Help  2 Inwrap  3  4 Hex  5  6  7 Search  8  9  10 Quit  11  12 Encoding
```

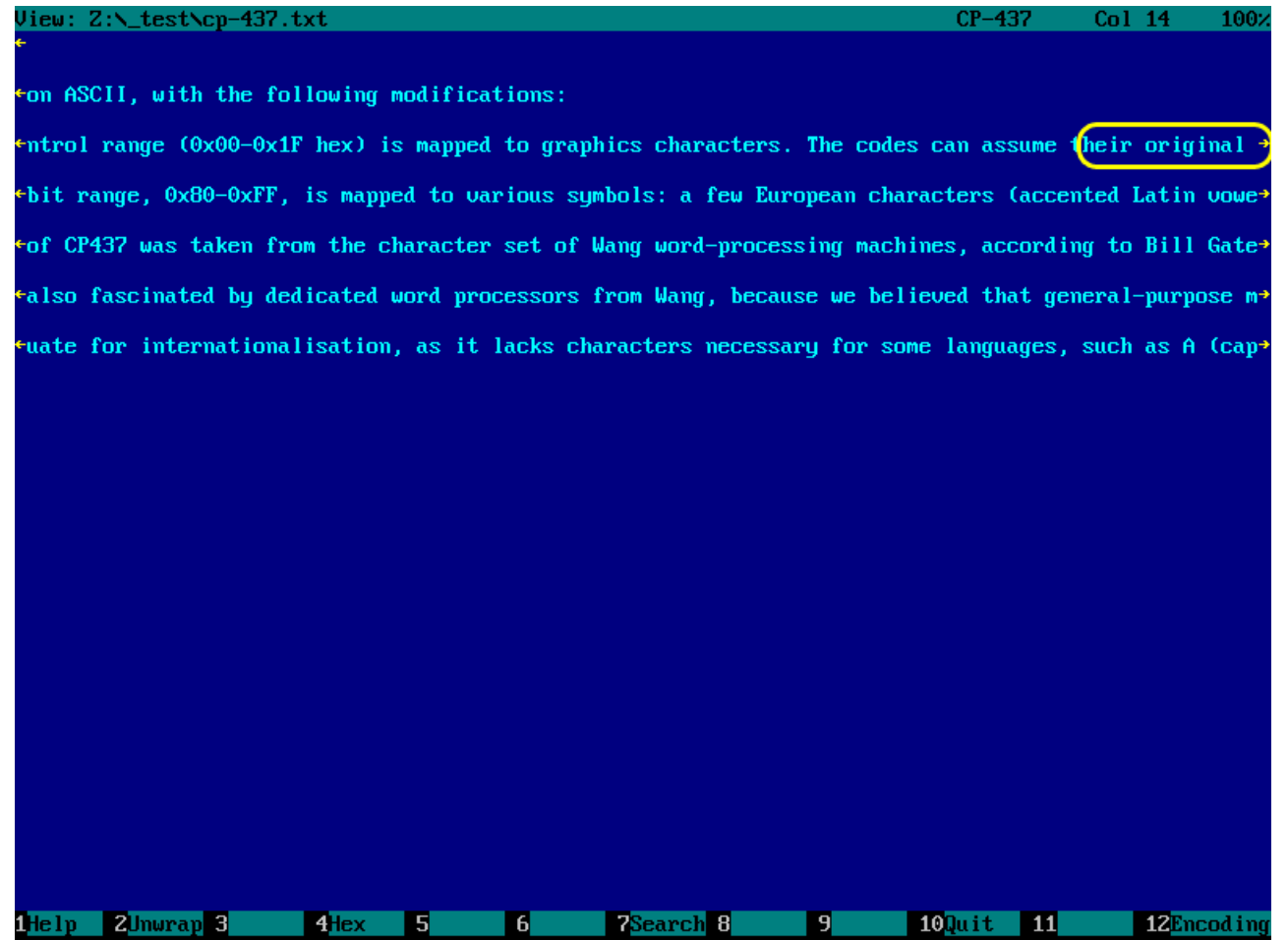
When line wrapping is off, lines which do not fit the screen will be marked with yellow arrows (←, →).



```
View: Z:\_test\cp-437.txt          CP-437    Col 0    100%
Code page 437
CP437 is based on ASCII, with the following modifications:
    * The C0 control range (0x00-0x1F hex) is mapped to graphics characters. The codes can assume f
    * The high-bit range, 0x80-0xFF, is mapped to various symbols: a few European characters (accen
The repertoire of CP437 was taken from the character set of Wang word-processing machines, accordin
    ": we were also fascinated by dedicated word processors from Wang, because we believed that gen
CP437 is inadequate for internationalisation, as it lacks characters necessary for some languages, →

1 Help  2 Unwrap  3          4 Hex  5          6          7 Search  8          9          10 Quit  11          12 Encoding
```

Use ← and → keys for horizontal scrolling (to see off-screen text).

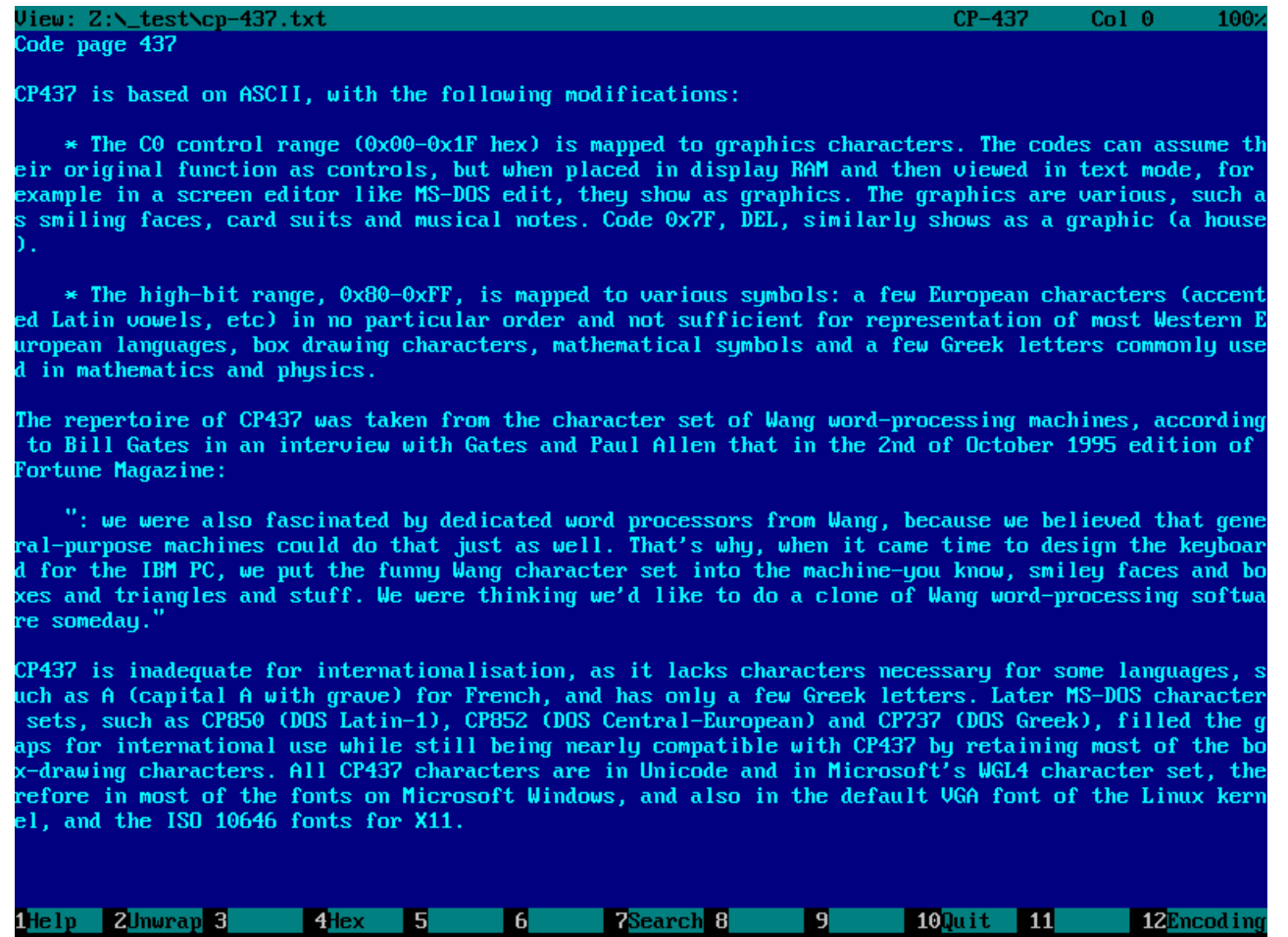


```
View: Z:\_test\cp-437.txt          CP-437    Col 14    100%
←
on ASCII, with the following modifications:
ontrol range (0x00-0x1F hex) is mapped to graphics characters. The codes can assume their original
bit range, 0x80-0xFF, is mapped to various symbols: a few European characters (accented Latin vowe
of CP437 was taken from the character set of Wang word-processing machines, according to Bill Gate
also fascinated by dedicated word processors from Wang, because we believed that general-purpose m
uate for internationalisation, as it lacks characters necessary for some languages, such as A (cap

1 Help  2 Unwrap  3  4 Hex  5  6  7 Search  8  9  10 Quit  11  12 Encoding
```

If you want to see full text of all long lines in the file without horizontal scrolling, enable line wrapping using the **F2** key.

This screenshot was taken with line wrapping enabled.



```
View: Z:\_test\cp-437.txt          CP-437    Col 0    100%
Code page 437

CP437 is based on ASCII, with the following modifications:

    * The C0 control range (0x00-0x1F hex) is mapped to graphics characters. The codes can assume their original function as controls, but when placed in display RAM and then viewed in text mode, for example in a screen editor like MS-DOS edit, they show as graphics. The graphics are various, such as smiling faces, card suits and musical notes. Code 0x7F, DEL, similarly shows as a graphic (a house ).

    * The high-bit range, 0x80-0xFF, is mapped to various symbols: a few European characters (accented Latin vowels, etc) in no particular order and not sufficient for representation of most Western European languages, box drawing characters, mathematical symbols and a few Greek letters commonly used in mathematics and physics.

The repertoire of CP437 was taken from the character set of Wang word-processing machines, according to Bill Gates in an interview with Gates and Paul Allen that in the 2nd of October 1995 edition of Fortune Magazine:

    " : we were also fascinated by dedicated word processors from Wang, because we believed that general-purpose machines could do that just as well. That's why, when it came time to design the keyboard for the IBM PC, we put the funny Wang character set into the machine—you know, smiley faces and boxes and triangles and stuff. We were thinking we'd like to do a clone of Wang word-processing software someday."

CP437 is inadequate for internationalisation, as it lacks characters necessary for some languages, such as Å (capital A with grave) for French, and has only a few Greek letters. Later MS-DOS character sets, such as CP850 (DOS Latin-1), CP852 (DOS Central-European) and CP737 (DOS Greek), filled the gaps for international use while still being nearly compatible with CP437 by retaining most of the box-drawing characters. All CP437 characters are in Unicode and in Microsoft's WGL4 character set, therefore in most of the fonts on Microsoft Windows, and also in the default VGA font of the Linux kernel, and the ISO 10646 fonts for X11.

1 | help | 2 | Unwrap | 3 | | 4 | hex | 5 | | 6 | | 7 | Search | 8 | | 9 | | 10 | Quit | 11 | | 12 | Encoding
```



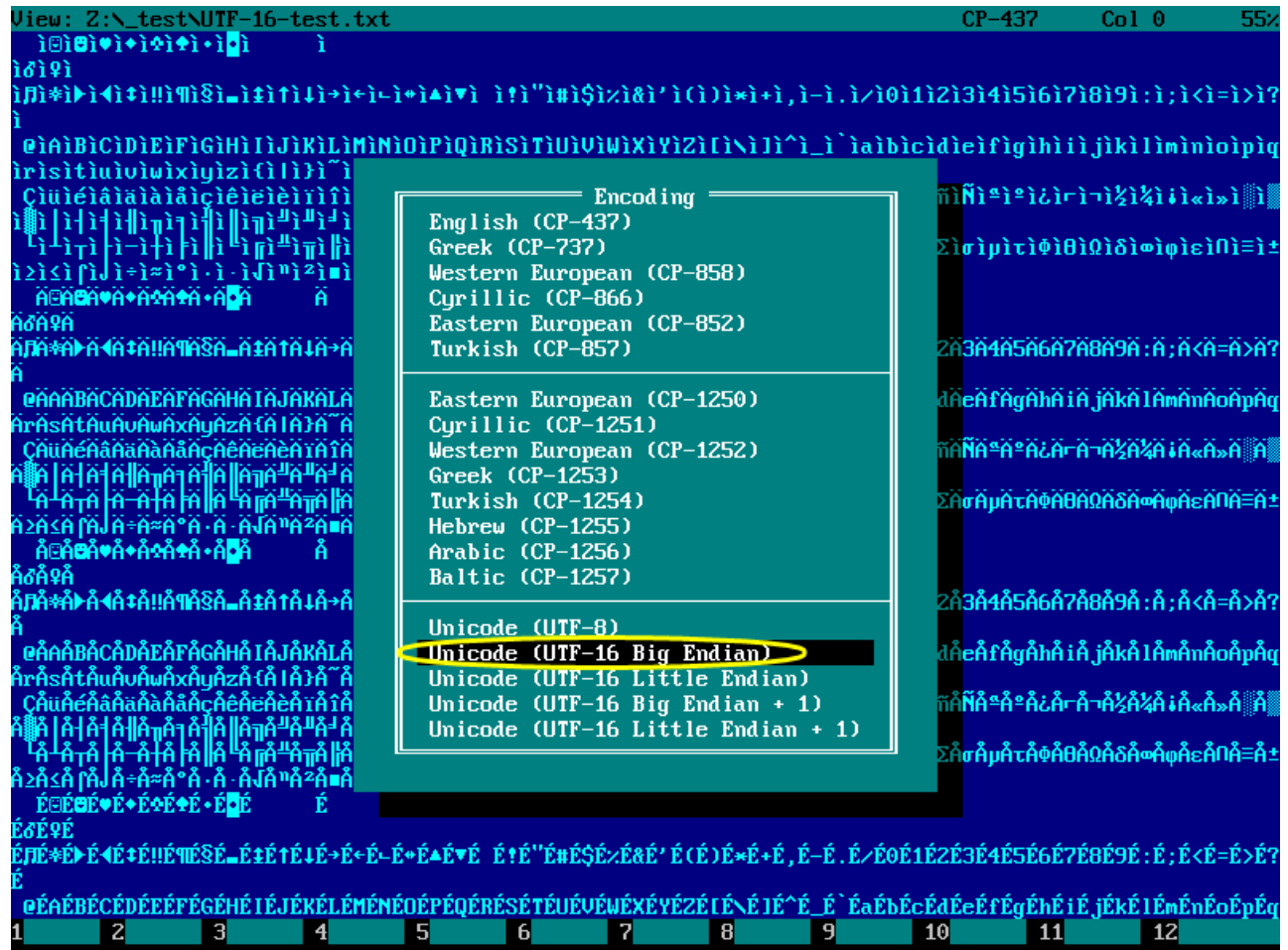
### 3.4. How to change encoding

To change encoding, press the **F12** key.

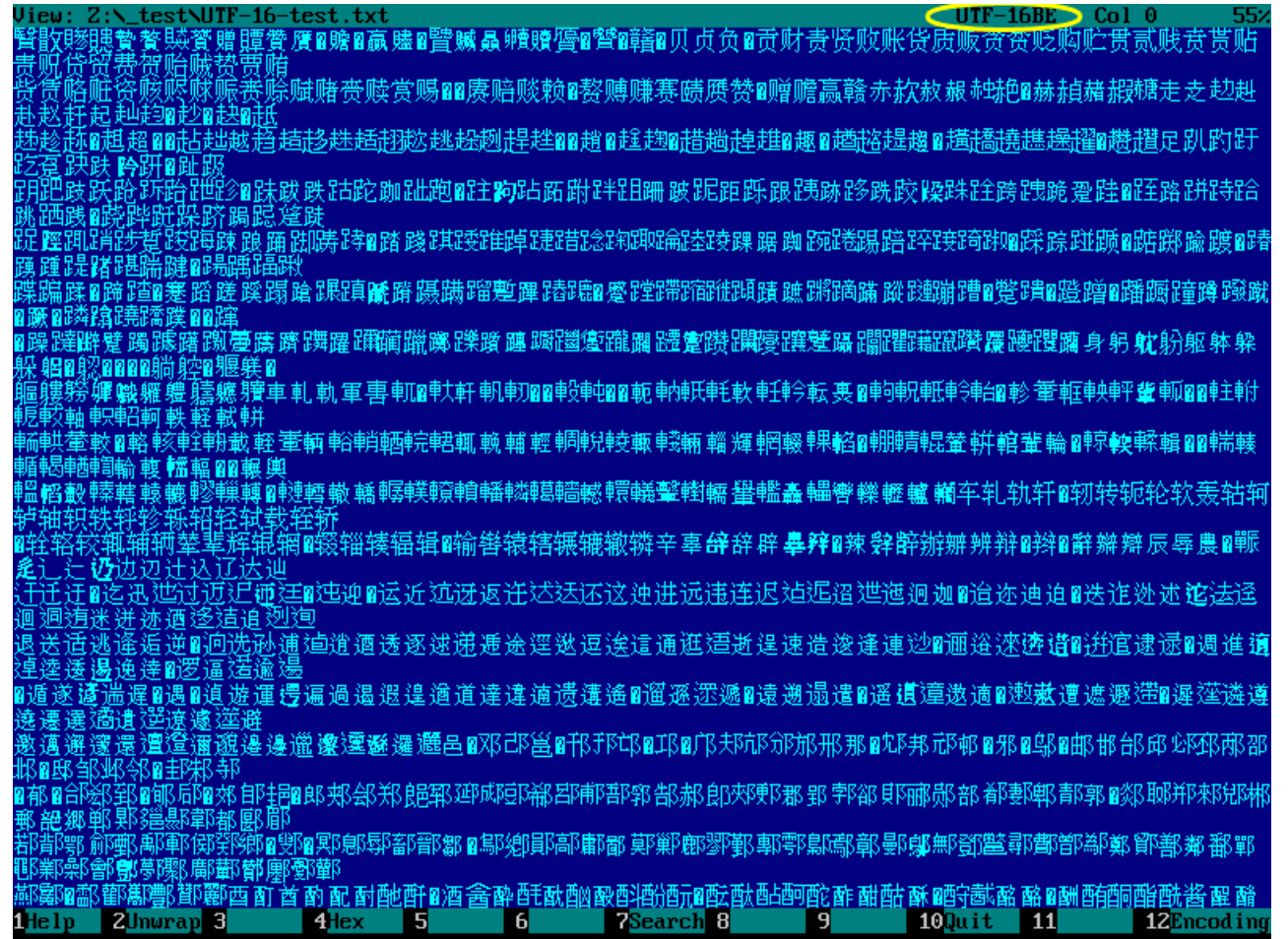


A popup window with a list of all encodings will be shown.

Choose new encoding using ↑ and ↓ keys followed by **Enter**; or left-click it with a mouse.



This screenshot shows text after encoding is changed.



### 3.5. How to search for a text in the file

To search for a text in the file, press the **F7** key.

```
View: Z:\_test\UTF-8-test.txt          CP-437   Col 0   0%
UTF-8 decoder capability and stress test
-----
Markus Kuhn <http://www.cl.cam.ac.uk/~mgk25/> - 2003-02-19

This test file can help you examine, how your UTF-8 decoder handles
various types of correct, malformed, or otherwise interesting UTF-8
sequences. This file is not meant to be a conformance test. It does
not prescribe any particular outcome and therefore there is no way to
"pass" or "fail" this test file, even though the text suggests a
preferable decoder behaviour at some places. The aim is instead to
help you think about and test the behaviour of your UTF-8 on a
systematic collection of unusual inputs. Experience so far suggests
that most first-time authors of UTF-8 decoders find at least one
serious problem in their decoder by using this file.

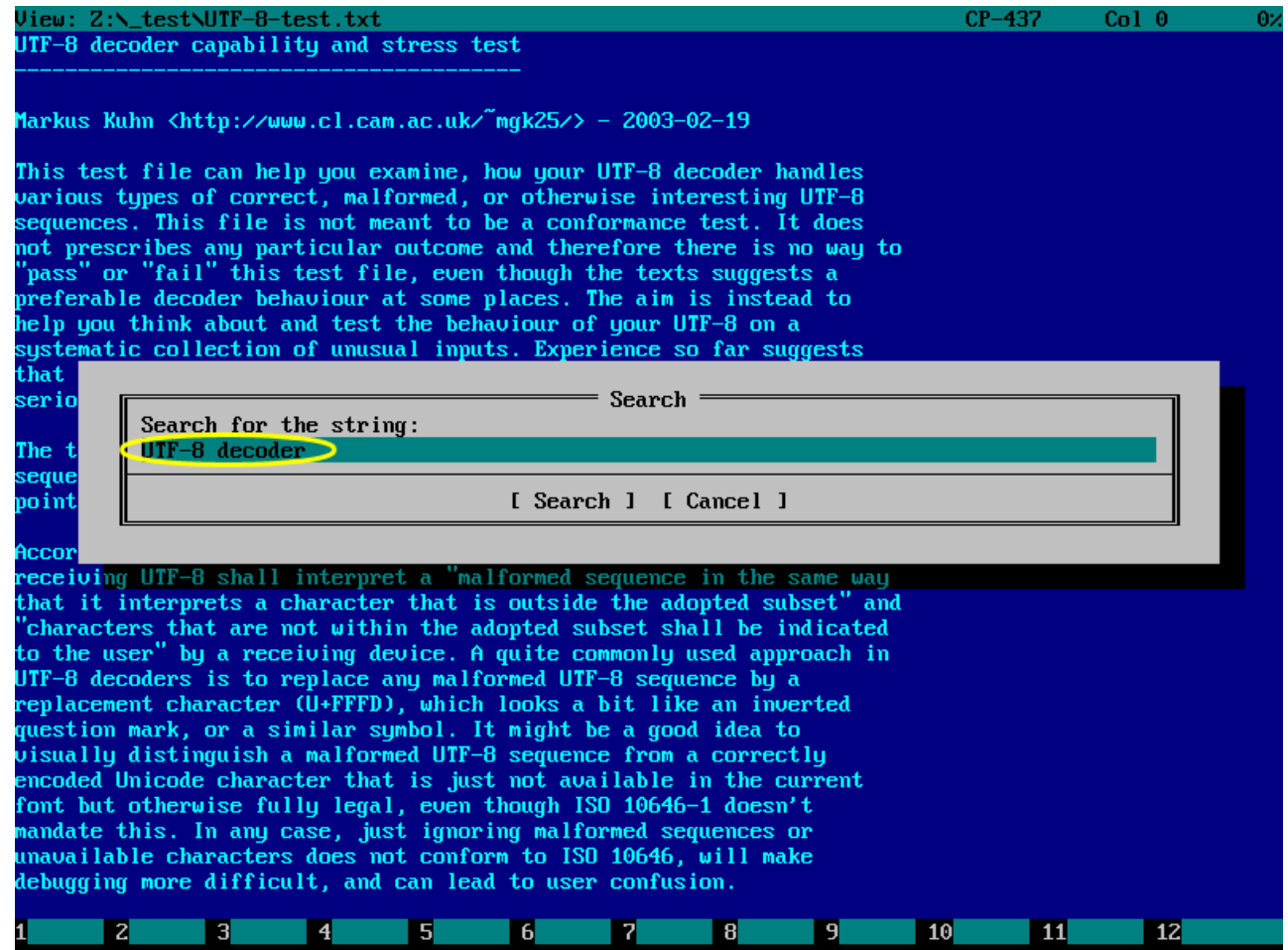
The test lines below cover boundary conditions, malformed UTF-8
sequences as well as correctly encoded UTF-8 sequences of Unicode code
points that should never occur in a correct UTF-8 file.

According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving device. A quite commonly used approach in
UTF-8 decoders is to replace any malformed UTF-8 sequence by a
replacement character (U+FFFD), which looks a bit like an inverted
question mark, or a similar symbol. It might be a good idea to
visually distinguish a malformed UTF-8 sequence from a correctly
encoded Unicode character that is just not available in the current
font but otherwise fully legal, even though ISO 10646-1 doesn't
mandate this. In any case, just ignoring malformed sequences or
unavailable characters does not conform to ISO 10646, will make
debugging more difficult, and can lead to user confusion.

1 Help  2 Unwrap  3  4 Hex  5  6  7 Search  8  9  10 Quit  11  12 Encoding
```

Enter a text to be searched into the popup window.

Then start the search by pressing the **Enter** key or clicking the *Search* button.



The screenshot shows a terminal window with a blue background and white text. The title bar at the top reads "View: Z:\\_test\UTF-8-test.txt CP-437 Col 0 0%". The main text in the terminal is the title "UTF-8 decoder capability and stress test" followed by a dashed line and the author information "Markus Kuhn <http://www.cl.cam.ac.uk/~mgk25/> - 2003-02-19". The main body of text discusses UTF-8 decoder behavior. A search dialog box is overlaid on the text, with the text "UTF-8 decoder" entered into the search field and highlighted with a yellow oval. The dialog box has a "Search" button and a "Cancel" button. At the bottom of the terminal window, there is a row of numbers from 1 to 12, each in a small box.

```
View: Z:\_test\UTF-8-test.txt CP-437 Col 0 0%
UTF-8 decoder capability and stress test
-----
Markus Kuhn <http://www.cl.cam.ac.uk/~mgk25/> - 2003-02-19

This test file can help you examine, how your UTF-8 decoder handles
various types of correct, malformed, or otherwise interesting UTF-8
sequences. This file is not meant to be a conformance test. It does
not prescribes any particular outcome and therefore there is no way to
"pass" or "fail" this test file, even though the texts suggests a
preferable decoder behaviour at some places. The aim is instead to
help you think about and test the behaviour of your UTF-8 on a
systematic collection of unusual inputs. Experience so far suggests
that
serio

The t
seque
point

Accor
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving device. A quite commonly used approach in
UTF-8 decoders is to replace any malformed UTF-8 sequence by a
replacement character (U+FFFD), which looks a bit like an inverted
question mark, or a similar symbol. It might be a good idea to
visually distinguish a malformed UTF-8 sequence from a correctly
encoded Unicode character that is just not available in the current
font but otherwise fully legal, even though ISO 10646-1 doesn't
mandate this. In any case, just ignoring malformed sequences or
unavailable characters does not conform to ISO 10646, will make
debugging more difficult, and can lead to user confusion.

1 2 3 4 5 6 7 8 9 10 11 12
```

Text found, if any, will be highlighted.

```
View: 2:~ test\UTF-8-test.txt CP-437 Col 0 0%
UTF-8 decoder capability and stress test
-----
Markus Kuhn <http://www.cl.cam.ac.uk/~mgk25/> - 2003-02-19

This test file can help you examine, how your UTF-8 decoder handles
various types of correct, malformed, or otherwise interesting UTF-8
sequences. This file is not meant to be a conformance test. It does
not prescribe any particular outcome and therefore there is no way to
"pass" or "fail" this test file, even though the text suggests a
preferable decoder behaviour at some places. The aim is instead to
help you think about and test the behaviour of your UTF-8 on a
systematic collection of unusual inputs. Experience so far suggests
that most first-time authors of UTF-8 decoders find at least one
serious problem in their decoder by using this file.

The test lines below cover boundary conditions, malformed UTF-8
sequences as well as correctly encoded UTF-8 sequences of Unicode code
points that should never occur in a correct UTF-8 file.

According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving device. A quite commonly used approach in
UTF-8 decoders is to replace any malformed UTF-8 sequence by a
replacement character (U+FFFD), which looks a bit like an inverted
question mark, or a similar symbol. It might be a good idea to
visually distinguish a malformed UTF-8 sequence from a correctly
encoded Unicode character that is just not available in the current
font but otherwise fully legal, even though ISO 10646-1 doesn't
mandate this. In any case, just ignoring malformed sequences or
unavailable characters does not conform to ISO 10646, will make
debugging more difficult, and can lead to user confusion.

1 Help 2 Unwrap 3 4 Hex 5 6 7 Search 8 9 10 Quit 11 12 Encoding
```

To continue searching for the same text, press **Shift+F7**.

```
View: Z:\_test\UTF-8-test.txt CP-437 Col 0 0%
This test file can help you examine, how your UTF-8 decoder handles
various types of correct, malformed, or otherwise interesting UTF-8
sequences. This file is not meant to be a conformance test. It does
not prescribes any particular outcome and therefore there is no way to
"pass" or "fail" this test file, even though the texts suggests a
preferable decoder behaviour at some places. The aim is instead to
help you think about and test the behaviour of your UTF-8 on a
systematic collection of unusual inputs. Experience so far suggests
that most first-time authors of UTF-8 decoders find at least one
serious problem in their decoder by using this file.

The test lines below cover boundary conditions, malformed UTF-8
sequences as well as correctly encoded UTF-8 sequences of Unicode code
points that should never occur in a correct UTF-8 file.

According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving device. A quite commonly used approach in
UTF-8 decoders is to replace any malformed UTF-8 sequence by a
replacement character (U+FFFD), which looks a bit like an inverted
question mark, or a similar symbol. It might be a good idea to
visually distinguish a malformed UTF-8 sequence from a correctly
encoded Unicode character that is just not available in the current
font but otherwise fully legal, even though ISO 10646-1 doesn't
mandate this. In any case, just ignoring malformed sequences or
unavailable characters does not conform to ISO 10646, will make
debugging more difficult, and can lead to user confusion.

Please check, whether a malformed UTF-8 sequence is (1) represented at
all, (2) represented by exactly one single replacement character (or
equivalent signal), and (3) the following quotation mark after an
illegal UTF-8 sequence is correctly displayed, i.e. proper
resynchronization takes place immediately after any malformed
1|help 2|Unwrap 3| 4|hex 5| 6| 7|Search 8| 9| 10|Quit 11| 12|Encoding
```