

Edit Files

Contents	
1. How to start	2
2. Screen contents	3
3. How to	4
3.1. Exit editor	4
3.2. Save file	5
3.3. Navigation	6
3.4. Text insertion and removal	7
3.5. Undo changes	8
3.6. Change encoding	12
3.7. Delete, copy or move a text range	15
3.7. Delete, copy or move a rectangular block	24

1. How to start

To edit a file, set cursor over it using ↑ and ↓ keys or by clicking it with left mouse button. Then press the **F4** 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

_name      11:10

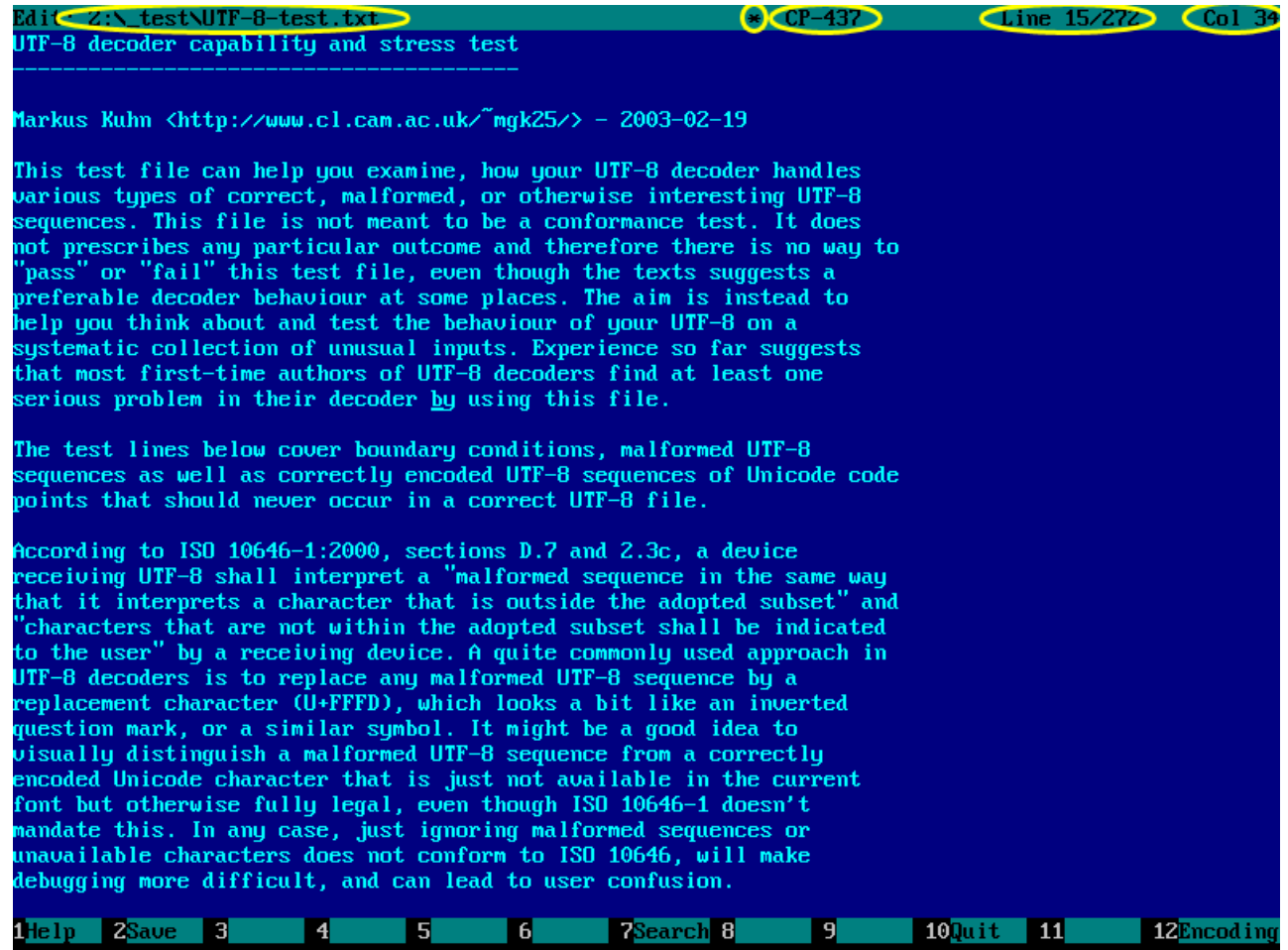
_test      Folder 05/11/08 10:55
UTF-8-test.txt 20334 05/11/08 10:55

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

2. Screen contents

Topmost line of the editor contains:

- name of file being edited
- asterisk (*), if file was changed since last save
- encoding type
- number of current line, slash, total number of lines in current file
- number of current column



The screenshot shows a text editor window with a dark blue background and light blue text. The title bar at the top is green and contains the following information: "Edi: z:\N test\UTF-8-test.txt" (with "z:\N test\UTF-8-test.txt" circled in yellow), an asterisk "*" (with "*" circled in yellow), "CP-437" (with "CP-437" circled in yellow), "Line 15/272" (with "Line 15/272" circled in yellow), and "Col 34" (with "Col 34" circled in yellow). The main content area displays the following text:

```
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.
```

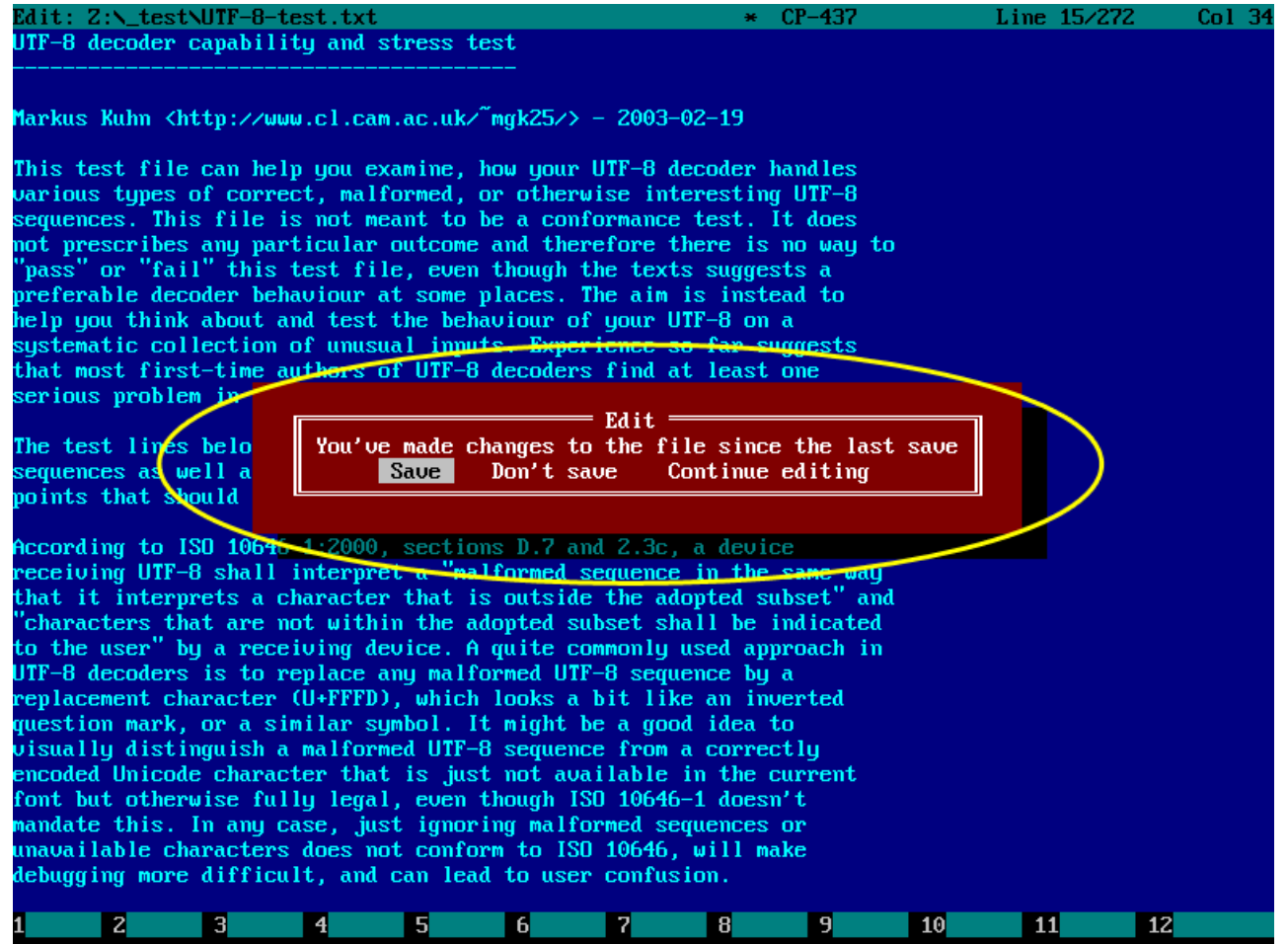
At the bottom of the window, there is a menu bar with the following items: 1help, 2Save, 3, 4, 5, 6, 7Search, 8, 9, 10Quit, 11, 12Encoding.

3. How to

3.1. Exit editor

To exit editor, press **F10** or **Esc** key.

If file was changed, you will be prompted to either *Save* it, to exit without saving (*Don't Save*) or to *Continue Editing* (editor will not be closed).



The screenshot shows a text editor window titled 'Edit: Z:_test\UTF-8-test.txt' with a status bar indicating '* CP-437 Line 15/272 Col 34'. The file content is a UTF-8 decoder stress test. A red dialog box with a white border is overlaid on the text, titled 'Edit' and containing the message: 'You've made changes to the file since the last save'. Below the message are three buttons: 'Save', 'Don't save', and 'Continue editing'. A yellow oval highlights the dialog box. The text in the background includes: '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...', 'The test lines below sequences as well as points that should', and '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.'

3.2. Save file

To save a file, press **F2**.

```
Edit: Z:\_test\UTF-8-test.txt * CP-437 Line 15/272 Col 34
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 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 Save 3 4 5 6 7 Search 8 9 10 Quit 11 12 Encoding
```

3.3. Navigation

To move text cursor within a file, use the following keys:

Operation	Key
Move cursor to the previous line	↑
Move cursor to the next line	↓
Move cursor to the previous character	←
Move cursor to the next character	→
Move cursor to the start of line	Home
Move cursor to the end of line	End
Move cursor to the first line in file	Ctrl+PageUp or Ctrl+Home
Move cursor to the last line in file	Ctrl+PageDown or Ctrl+End
Scroll view to the next page	Page Up
Scroll view to the previous page	Page Down

3.4. Text insertion and removal

To insert text, use letter keys, numeric keys and other usual keys.

To start a new line, use **Enter** key.

To delete text, use the following keys:

Operation	Key
Delete character under the cursor or join the next line	Delete
Delete character before the cursor or join to the previous line	Backspace
Delete the whole current line	Ctrl+Y

3.5. Undo changes

To undo last change, press **Ctrl+Z** or **Alt+Backspace**.

To redo last change, press **Ctrl+Shift+Z**.

Example: cursor is pointing to line 4.

```
Edit: Z:\_test\UTF-8-test.txt UTF-8 Line 4/72 Col 1
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 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|Save 3 4 5 6 7|Search 8 9 10|Quit 11 12|Encoding
```


Then you have deleted this line by pressing **Ctrl+Y**.

```
Edit: Z:\_test\UTF-8-test.txt * UTF-8 Line 4/271 Col 1
UTF-8 decoder capability and stress test
-----

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

To undo last change (line deleting in this example),
press **Ctrl+Z**.

```
Edit: Z:\_test\UTF-8-test.txt          UTF-8          Line 4/272          Col 1
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|save  3|  4|  5|  6|  7|search  8|  9| 10|quit  11| 12|encoding
```

To redo last change (e.g. delete line again), press **Ctrl+Shift+Z**.

```
Edit: Z:\_test\UTF-8-test.txt * UTF-8 Line 4/271 Col 1
UTF-8 decoder capability and stress test
-----

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 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.

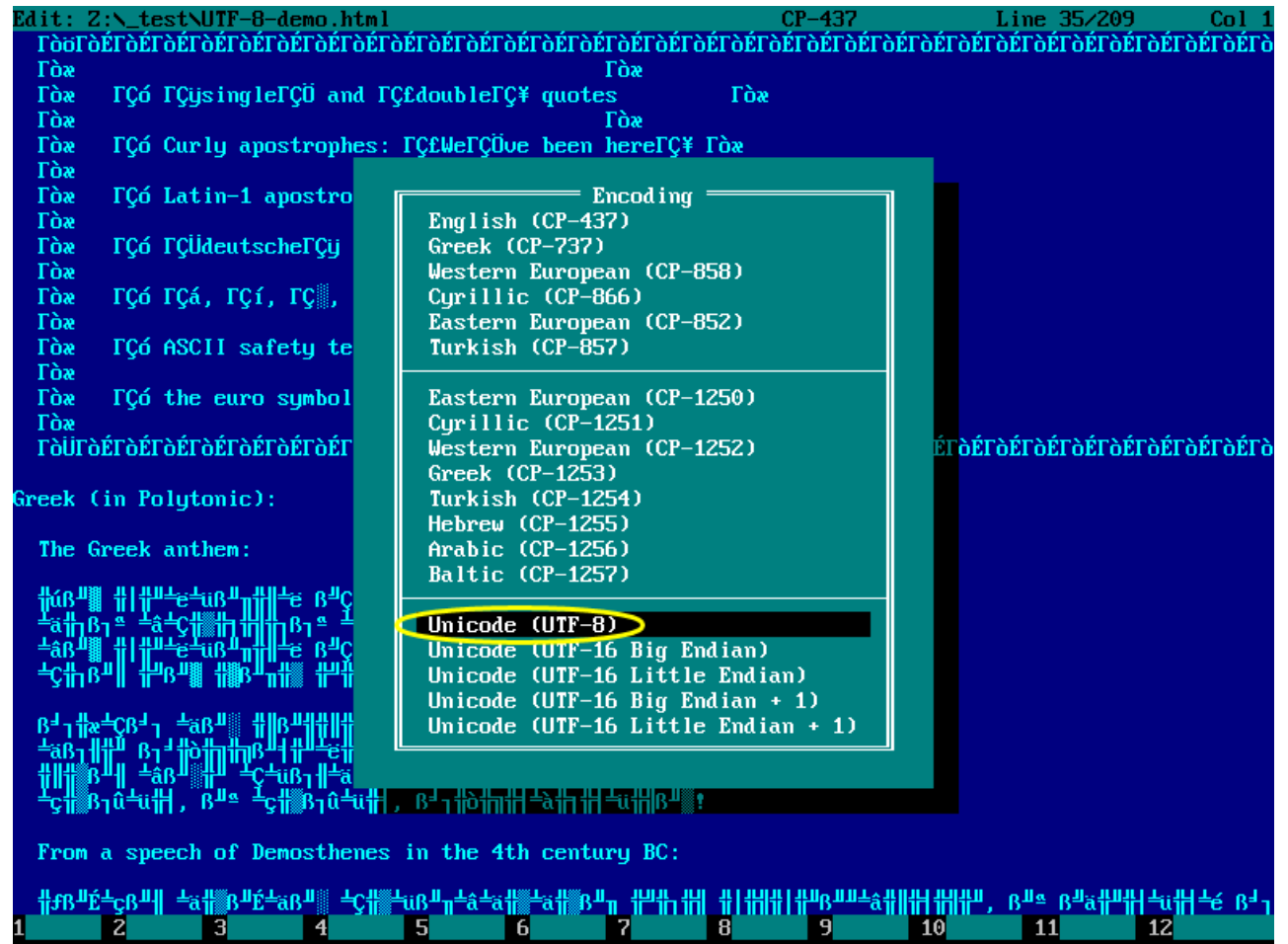
The test lines below cover boundary conditions, malformed UTF-8
sequences as well as correctly encoded UTF-8 sequences of Unicode code
points that should never occur in a correct UTF-8 file.

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.

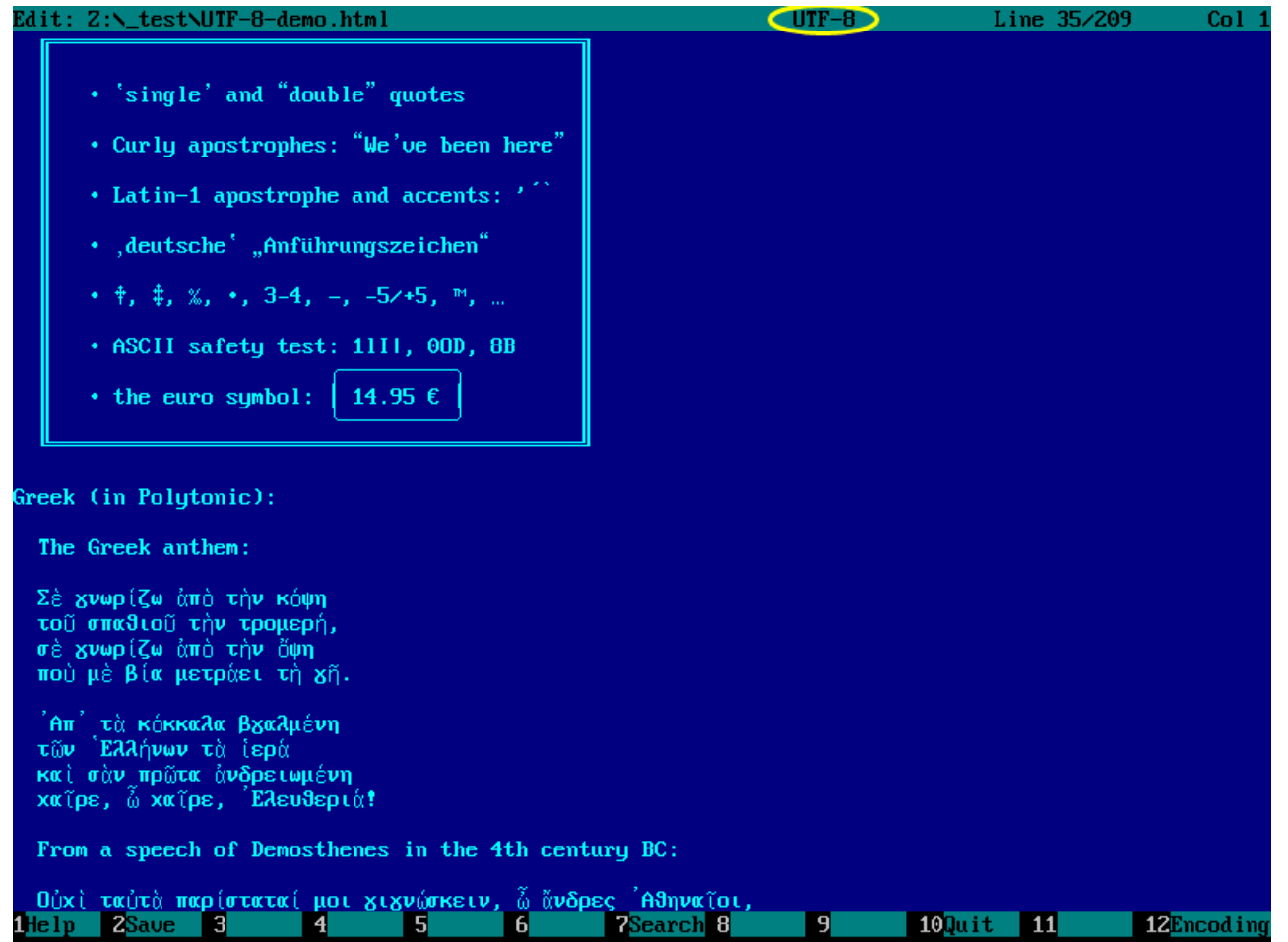
Please check, whether a malformed UTF-8 sequence is (1) represented at
1|help 2|Save 3| 4| 5| 6| 7|Search 8| 9| 10|Quit 11| 12|Encoding
```


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.



Screenshot shows the same file after encoding changed.



```
Edit: Z:\_test\UTF-8-demo.html UTF-8 Line 35/209 Col 1
• 'single' and "double" quotes
• Curly apostrophes: "We've been here"
• Latin-1 apostrophe and accents: ' ^
• ,deutsche' „Anführungszeichen“
• †, ‡, %, •, 3-4, -, -5/+5, ™, ...
• ASCII safety test: 1111, 000, 88
• the euro symbol: 14.95 €

Greek (in Polytonic):

The Greek anthem:

Σὲ γνωρίζω ἀπὸ τὴν κόψη
τοῦ σπαθιοῦ τὴν τρομερὴ,
σὲ γνωρίζω ἀπὸ τὴν ὄψη
ποῦ μὲ βία μετράει τὴ γῆ.

Ἄπ' τὰ κόκκαλα βγαλμένη
τῶν Ἑλλήνων τὸ ἱερὸ
καὶ σὸν πρῶτα ἀνδρειωμένη
καίρε, ὦ καίρε, Ἐλευθεριά!

From a speech of Demosthenes in the 4th century BC:

Οὐχὶ ταῦτ' ἀπορίσταταί μοι γιγνώσκεις, ὦ ἄνδρες Ἀθηναῖοι,
```

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

3.7. Delete, copy or move a text range

Text range is a continuous part of text which can be a substring or occupy multiple strings, either complete or incomplete.

Text range	Not a text range
Code page CP437 is based on ASCII, with the following modifications	Code page CP437 is based on ASCII, with the following modifications

Before text range can be deleted, copied or moved, it must be selected.

Select a text range

To select a text range, use following keys:

Operation	Key
Select text upward from the cursor position	Shift+↑
Select text downward from the cursor position	Shift+↓
Select text leftward from the cursor position	Shift+←
Select text rightward from the cursor position	Shift+→
Select text from the cursor to the start of current line	Shift+Home
Select text from the cursor to the end of current line	Shift+End
Select text from the cursor to the start of file	Ctrl+Shift+Home
Select text from the cursor to the end of file	Ctrl+Shift+End

Screenshot shows an example of selected text range.

```

Edit: Z:\_test\UTF-8-test.txt          CP-437          Line 21/272          Col 1
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 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.

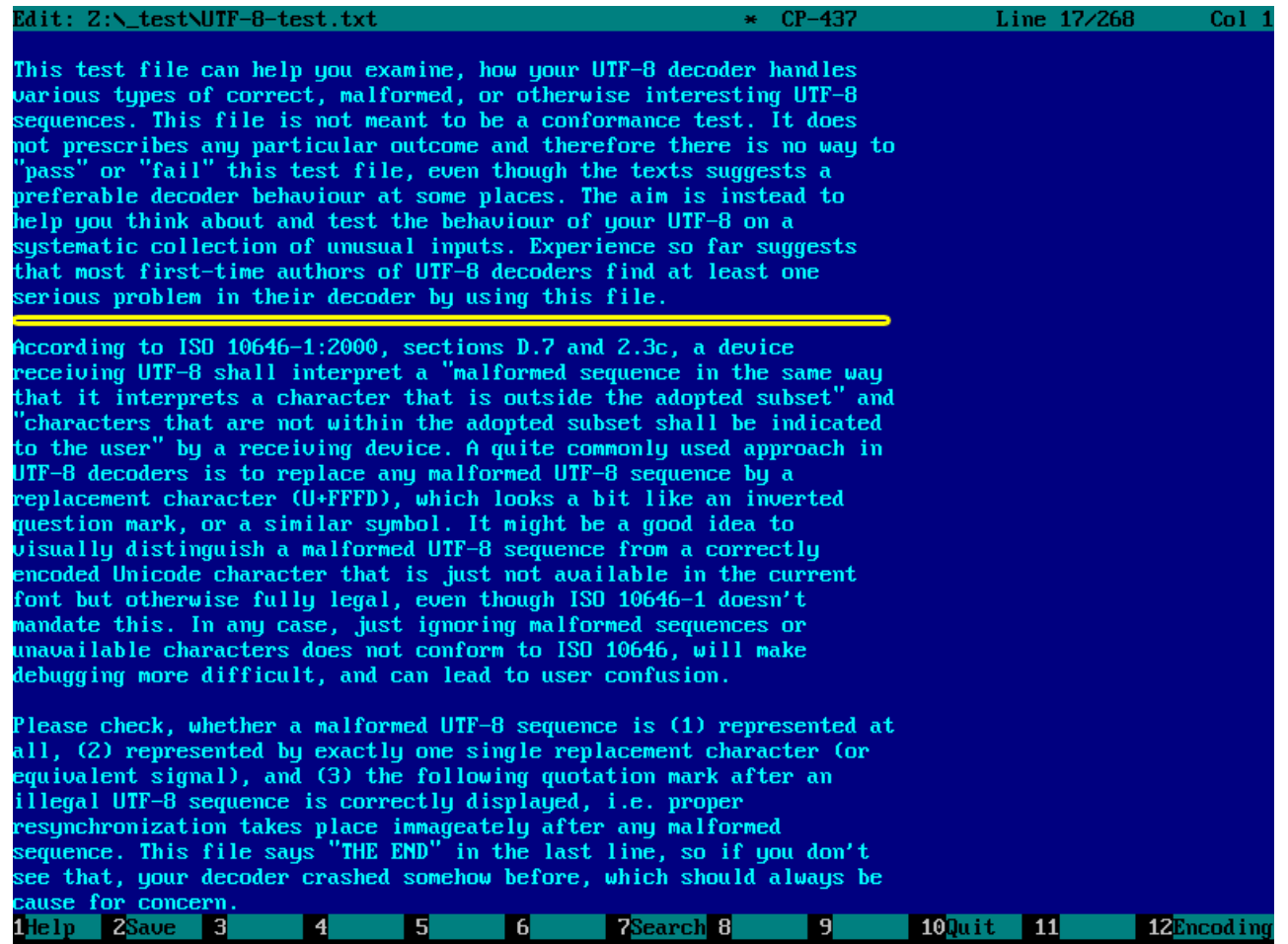
Please check, whether a malformed UTF-8 sequence is (1) represented at
all, (2) represented by exactly one single replacement character (or
equivalent signal), and (3) the following quotation mark after an
illegal UTF-8 sequence is correctly displayed, i.e. proper
1Help  2Save  3      4      5      6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

```


Delete a text range

To delete a text range, select it (see above) and press **Ctrl+D**.

Screenshot shows location of recently deleted text range.



```
Edit: Z:\_test\UTF-8-test.txt * CP-437 Line 17/268 Col 1

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.

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.

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

Copy text range

Before text range can be copied, it must be selected (see above).

After text range is selected, press **Ctrl+C** or **Ctrl+Insert** to copy it to the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt          CP-437          Line 21/272          Col 1
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.

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

Move the cursor to position which is intended to be a start of inserted text, and then press **Ctrl+V** or **Shift+Insert** to insert text from clipboard.

Screenshot is taken before text insertion from the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt          CP-437          Line 36/272          Col 1

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.

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

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

Screenshot is taken after text insertion from the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt * CP-437 Line 40/276 Col 1
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.

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.

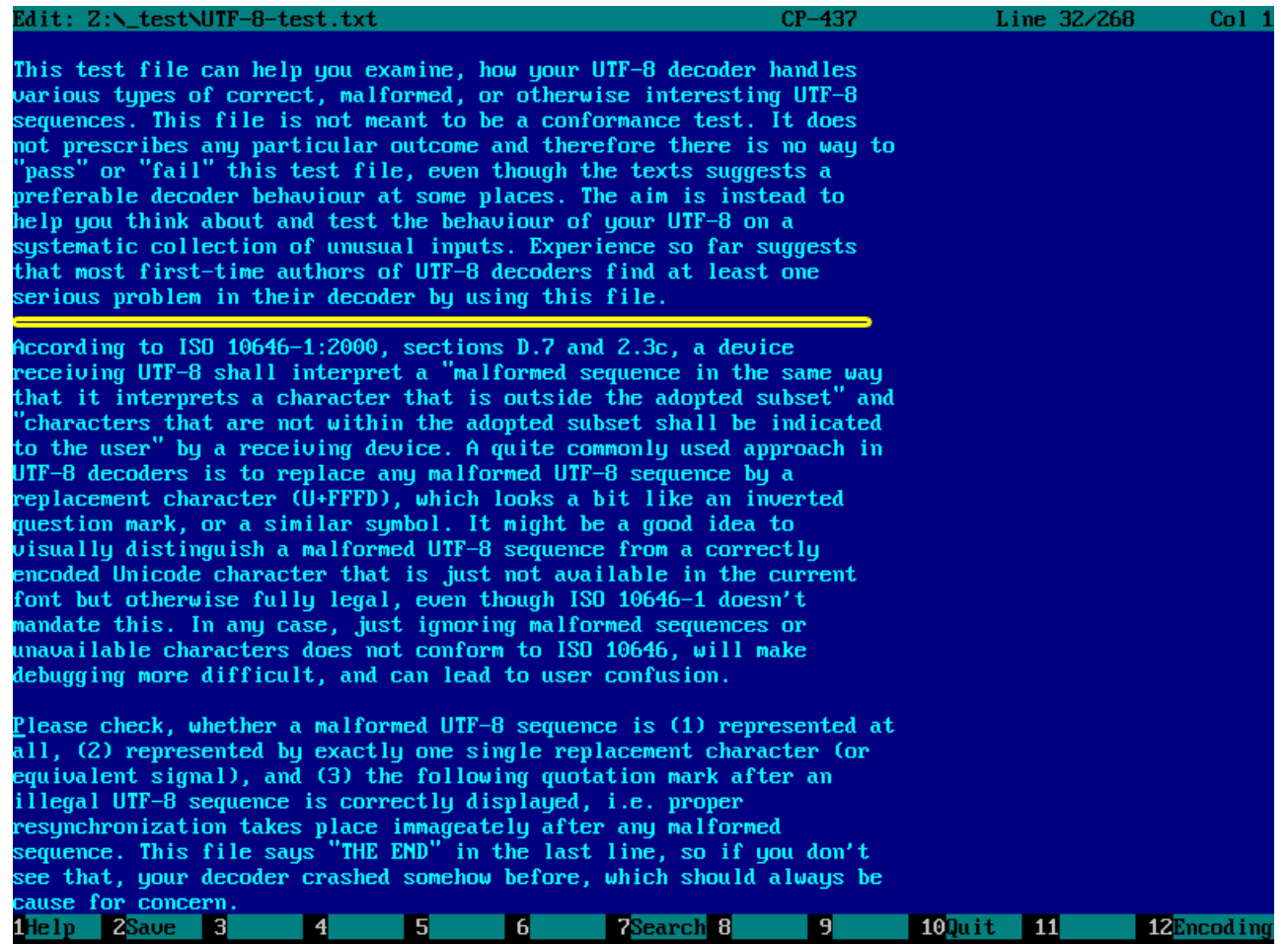
Please check, whether a malformed UTF-8 sequence is (1) represented at
1|help 2|Save 3| 4| 5| 6| 7|Search 8| 9| 10|Quit 11| 12|Encoding
```

Move text range

Before text range can be moved, it must be selected (see above).

After text range is selected, press **Ctrl+X** or **Shift+Delete** to move it to the clipboard.

Although a selected text range is instantly removed from the edited text, its copy is retained in the clipboard.



```
Edit: Z:\_test\UTF-8-test.txt          CP-437          Line 32/268          Col 1

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 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.

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.

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

Move the cursor to position which is intended to be a start of inserted text, and then press **Ctrl+V** or **Shift+Insert**.

Screenshot is taken before text insertion from the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt          CP-437          Line 32/268          Col 1

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 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.

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.

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

Screenshot is taken after text insertion from the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt * CP-437 Line 36/272 Col 1

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.

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.

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.

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

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

3.7. Delete, copy or move a rectangular block

Rectangular block is a set of substrings of multiple adjacent strings, cut in such way that starting and ending positions of all substrings are the same.

Rectangular block	Not a rectangular block
05 = U+0005 : ENQUIRY	05 = U+0005 : ENQUIRY
06 = U+0006 : ACKNOWLEDGE	06 = U+0006 : ACKNOWLEDGE
07 = U+0007 : BELL	07 = U+0007 : BELL
08 = U+0008 : BACKSPACE	08 = U+0008 : BACKSPACE

Before rectangular block can be deleted, copied or moved, it must be selected.

Select a rectangular block

To select a rectangular block, use following keys:

Operation	Key
Select rectangular block upward from the cursor position	Alt+Shift+↑
Select rectangular block downward from the cursor position	Alt+Shift+↓
Select rectangular block leftward from the cursor position	Alt+Shift+←
Select rectangular block rightward from the cursor position	Alt+Shift+→
Select rectangular block from the cursor to the start of current line	Alt+Shift+Home
Select rectangular block from the cursor to the end of current line	Alt+Shift+End
Select rectangular block from the cursor to the start of file	Alt+Ctrl+Shift+Home
Select rectangular block from the cursor to the end of file	Alt+Ctrl+Shift+End

Here is an example of selected rectangular block.

```
Edit: Z:\_test\UTF-8-test.txt UTF-8 Line 85/272 Col 15
1 Some correct UTF-8 text
You should see the Greek word 'kosme': "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000): " "
2.1.2 2 bytes (U-00000080): "  "
2.1.3 3 bytes (U-00000800): "   "
2.1.4 4 bytes (U-00010000): "    "
2.1.5 5 bytes (U-00200000): "     "
2.1.6 6 bytes (U-04000000): "      "
2.2 Last possible sequence of a certain length
2.2.1 1 byte (U-0000007F): " "
2.2.2 2 bytes (U-000007FF): "  "
2.2.3 3 bytes (U-0000FFFF): "   "
2.2.4 4 bytes (U-001FFFFFF): "    "
2.2.5 5 bytes (U-03FFFFFFF): "     "
2.2.6 6 bytes (U-7FFFFFFF): "      "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = "  "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "
3 Malformed sequences
3.1 Unexpected continuation bytes
1|help 2|Save 3| 4| 5| 6| 7|Search 8| 9| 10|Quit 11| 12|Encoding
```

Delete rectangular block

To delete rectangular block, select it (see above) and press **Ctrl+D**.

Screenshot shows former location of deleted rectangular block.

```
Edit: Z:\_test\UTF-8-test.txt * UTF-8 Line 80/272 Col 8
1 Some correct UTF-8 text
You should see the Greek word 'kosme': "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000): " "
2.1.2 2 bytes (U-00000080): "  "
2.1.3 3 bytes (U-00000800): "   "
2.1.4 4 bytes (U-00010000): "    "
2.1.5 5 bytes (U-00200000): "     "
2.1.6 6 bytes (U-04000000): "      "
2.2 Last possible sequence of a certain length
2.2.1 (U-0000007F): " "
2.2.2 (U-000007FF): "  "
2.2.3 (U-0000FFFF): "   "
2.2.4 (U-001FFFFFF): "    "
2.2.5 (U-03FFFFFF): "     "
2.2.6 (U-7FFFFFFF): "      "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = "   "
2.3.2 U-0000E000 = ee 80 80 = "   "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "
3 Malformed sequences
3.1 Unexpected continuation bytes
1 help 2 Save 3 4 5 6 7 Search 8 9 10 Quit 11 12 Encoding
```

Copy rectangular block

Before rectangular block can be copied, it must be selected (see above).

After rectangular block is selected, press **Ctrl+C** or **Ctrl+Insert** to copy it to the clipboard.

```
Edit: Z:\_test\UTF-8-test.txt          UTF-8          Line 85/272   Col 15
1 Some correct UTF-8 text
You should see the Greek word 'kosme':   "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000):   " "
2.1.2 2 bytes (U-00000080):  "  "
2.1.3 3 bytes (U-00000800):  "   "
2.1.4 4 bytes (U-00010000):  "    "
2.1.5 5 bytes (U-00200000):  "     "
2.1.6 6 bytes (U-04000000):  "      "
2.2 Last possible sequence of a certain length
2.2.1 1 byte (U-0000007F):   " "
2.2.2 2 bytes (U-000007FF):  "  "
2.2.3 3 bytes (U-0000FFFF):  "   "
2.2.4 4 bytes (U-001FFFFFF): "    "
2.2.5 5 bytes (U-03FFFFFF):  "     "
2.2.6 6 bytes (U-7FFFFFFF):  "      "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = "  "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "
3 Malformed sequences
3.1 Unexpected continuation bytes
1 help  2 Save  3  4  5  6  7 Search  8  9  10 Quit  11  12 Encoding
```

Move the cursor to position which is intended to be a start of inserted rectangular block, and then press **Ctrl+V** or **Shift+Insert** key to insert rectangular block from the clipboard.

Screenshot is taken before insertion of rectangular block from clipboard.

```

Edit: Z:\_test\UTF-8-test.txt          UTF-8          Line 80/272   Col 49
1 Some correct UTF-8 text
You should see the Greek word 'kosme':      "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000):      " "
2.1.2 2 bytes (U-00000080):     "  "
2.1.3 3 bytes (U-00000800):    "   "
2.1.4 4 bytes (U-00010000):    "    "
2.1.5 5 bytes (U-00200000):    "     "
2.1.6 6 bytes (U-04000000):    "      "
2.2 Last possible sequence of a certain length
2.2.1 1 byte (U-0000007F):     " "
2.2.2 2 bytes (U-000007FF):    "  "
2.2.3 3 bytes (U-0000FFFF):    "   "
2.2.4 4 bytes (U-001FFFFFF):   "    "
2.2.5 5 bytes (U-03FFFFFF):   "     "
2.2.6 6 bytes (U-7FFFFFFF):   "      "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = " "
2.3.3 U-0000FFFD = ef bf bd = " "
2.3.4 U-0010FFFF = f4 8f bf bf = " "
2.3.5 U-00110000 = f4 90 80 80 = " "
3 Malformed sequences
3.1 Unexpected continuation bytes
1|help  2|Save  3|  4|  5|  6|  7|Search  8|  9| 10|Quit  11| 12|Encoding

```

Screenshot is taken before insertion of rectangular block from the clipboard.

```

Edit: Z:\_test\UTF-8-test.txt * UTF-8 Line 80/272 Col 49
1 Some correct UTF-8 text

You should see the Greek word 'kosme': "κόσμη"

2 Boundary condition test cases

2.1 First possible sequence of a certain length

2.1.1 1 byte (U-00000000): " "
2.1.2 2 bytes (U-00000080): "  "
2.1.3 3 bytes (U-00000800): "   "
2.1.4 4 bytes (U-00010000): "    "
2.1.5 5 bytes (U-00200000): "     "
2.1.6 6 bytes (U-04000000): "      "

2.2 Last possible sequence of a certain length

2.2.1 1 byte (U-0000007F): " "
2.2.2 2 bytes (U-000007FF): "  "
2.2.3 3 bytes (U-0000FFFF): "   "
2.2.4 4 bytes (U-001FFFFFF): "    "
2.2.5 5 bytes (U-03FFFFFFF): "     "
2.2.6 6 bytes (U-7FFFFFFF): "      "

2.3 Other boundary conditions

2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = "  "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "

3 Malformed sequences

3.1 Unexpected continuation bytes

```

Move rectangular block

Before rectangular block can be moved, it must be selected (see above).

After rectangular block is selected, press **Ctrl+X** or **Shift+Delete** to move it to the clipboard.

Although selected rectangular block is instantly removed from the edited text, its copy is retained in the clipboard.

```

Edit: Z:\_test\UTF-8-test.txt          * UTF-8          Line 80/272          Col 8
1 Some correct UTF-8 text
You should see the Greek word 'kosme':      "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000):      " "
2.1.2 2 bytes (U-00000080):     "  "
2.1.3 3 bytes (U-00000800):    "   "
2.1.4 4 bytes (U-00010000):    "    "
2.1.5 5 bytes (U-00200000):    "     "
2.1.6 6 bytes (U-04000000):    "      "
2.2 Last possible sequence of a certain length
2.2.1 (U-0000007F):           " "
2.2.2 (U-000007FF):           "  "
2.2.3 (U-0000FFFF):           "   "
2.2.4 (U-001FFFFFF):          "    "
2.2.5 (U-03FFFFFF):           "     "
2.2.6 (U-7FFFFFFF):           "      "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = "  "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "
3 Malformed sequences
3.1 Unexpected continuation bytes
1 Help  2 Save  3  4  5  6  7 Search  8  9  10 Quit  11  12 Encoding

```

Move the cursor to position which is intended to be a start of inserted rectangular block, and then press **Ctrl+V** or **Shift+Insert** key to insert rectangular block from the clipboard.

Screenshot is taken before insertion of rectangular block from the clipboard.

```

Edit: Z:\_test\UTF-8-test.txt          * UTF-8          Line 80/272   Col 42
1 Some correct UTF-8 text
You should see the Greek word 'kosme':      "κόσμη"
2 Boundary condition test cases
2.1 First possible sequence of a certain length
2.1.1 1 byte (U-00000000):      " "
2.1.2 2 bytes (U-00000080):     "  "
2.1.3 3 bytes (U-00000800):     "   "
2.1.4 4 bytes (U-00010000):     "    "
2.1.5 5 bytes (U-00200000):     "     "
2.1.6 6 bytes (U-04000000):     "      "
2.2 Last possible sequence of a certain length
2.2.1 (U-0000007F):           "DEL"
2.2.2 (U-000007FF):           " "
2.2.3 (U-0000FFFF):           "  "
2.2.4 (U-001FFFFFF):          "   "
2.2.5 (U-03FFFFFF):           "    "
2.2.6 (U-7FFFFFFF):           "     "
2.3 Other boundary conditions
2.3.1 U-0000D7FF = ed 9f bf = "   "
2.3.2 U-0000E000 = ee 80 80 = "   "
2.3.3 U-0000FFFD = ef bf bd = "   "
2.3.4 U-0010FFFF = f4 8f bf bf = "    "
2.3.5 U-00110000 = f4 90 80 80 = "    "
3 Malformed sequences
3.1 Unexpected continuation bytes
1|help  2|Save  3|  4|  5|  6|  7|Search  8|  9| 10|Quit  11| 12|Encoding

```


Screenshot is taken after insertion of rectangular block from the clipboard.

```

Edit: Z:\_test\UTF-8-test.txt          * UTF-8          Line 80/272   Col 42
1 Some correct UTF-8 text

You should see the Greek word 'kosme':      "κόσμη"

2 Boundary condition test cases

2.1 First possible sequence of a certain length

2.1.1 1 byte (U-00000000):      " "
2.1.2 2 bytes (U-00000080):     "  "
2.1.3 3 bytes (U-00000800):    "   "
2.1.4 4 bytes (U-00010000):    "    "
2.1.5 5 bytes (U-00200000):    "     "
2.1.6 6 bytes (U-04000000):    "      "

2.2 Last possible sequence of a certain length

2.2.1 (U-0000007F):           "DEL"
2.2.2 (U-000007FF):           " "
2.2.3 (U-0000FFFF):           "  "
2.2.4 (U-001FFFFFF):          "   "
2.2.5 (U-03FFFFFF):           "    "
2.2.6 (U-7FFFFFFF):           "     "

2.3 Other boundary conditions

2.3.1 U-0000D7FF = ed 9f bf = " "
2.3.2 U-0000E000 = ee 80 80 = " "
2.3.3 U-0000FFFD = ef bf bd = " "
2.3.4 U-0010FFFF = f4 8f bf bf = " "
2.3.5 U-00110000 = f4 90 80 80 = " "

3 Malformed sequences

3.1 Unexpected continuation bytes

```

1 help 2 Save 3 4 5 6 7 Search 8 9 10 Quit 11 12 Encoding