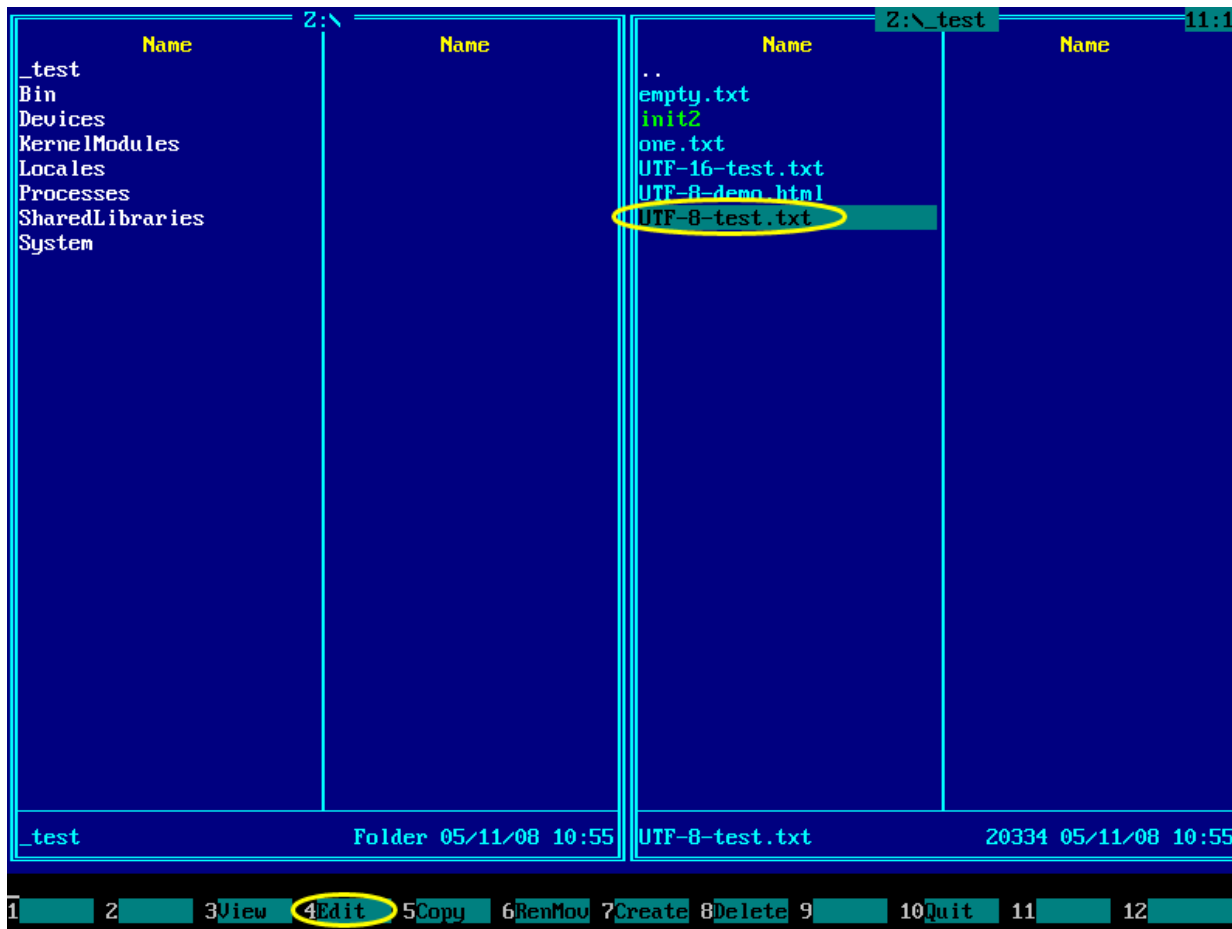

Edit Files

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. Scroll file	6
3.4. Input new text or delete text	7
3.5. Undo changes	8
3.6. Change encoding	12
3.7. Delete, copy, move block	15
3.8. Delete, copy, move rectangular block.....	25

1. How to start

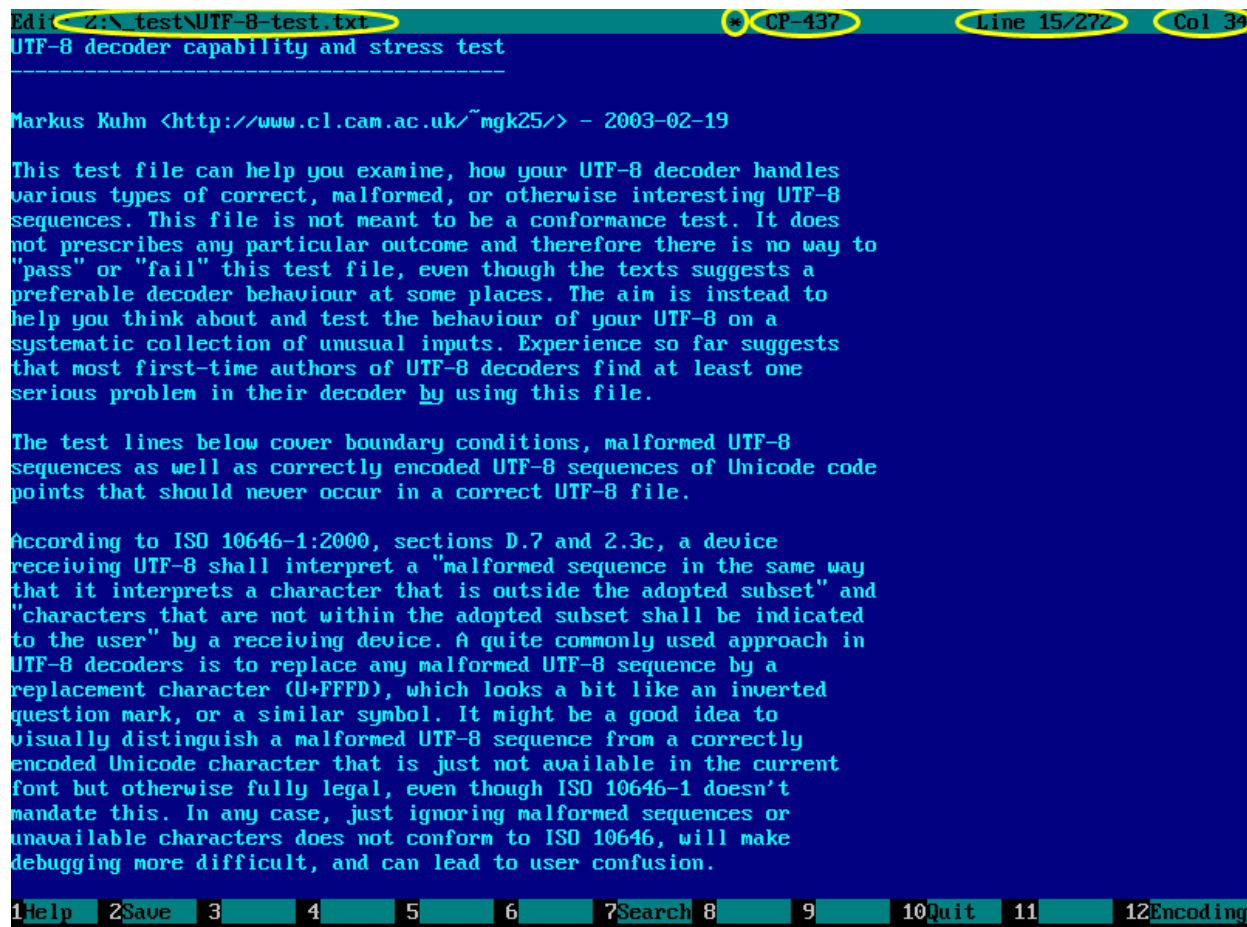
To edit file, set cursor over it using ↑ and ↓ keys or by clicking it with left mouse button and press **F4** key.



2. Screen contents

Top line of editor screen displays:

- 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



```
Edi Z:\ testUTF-8-test.txt CP-437 Line 15/27z Col 3z
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 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.

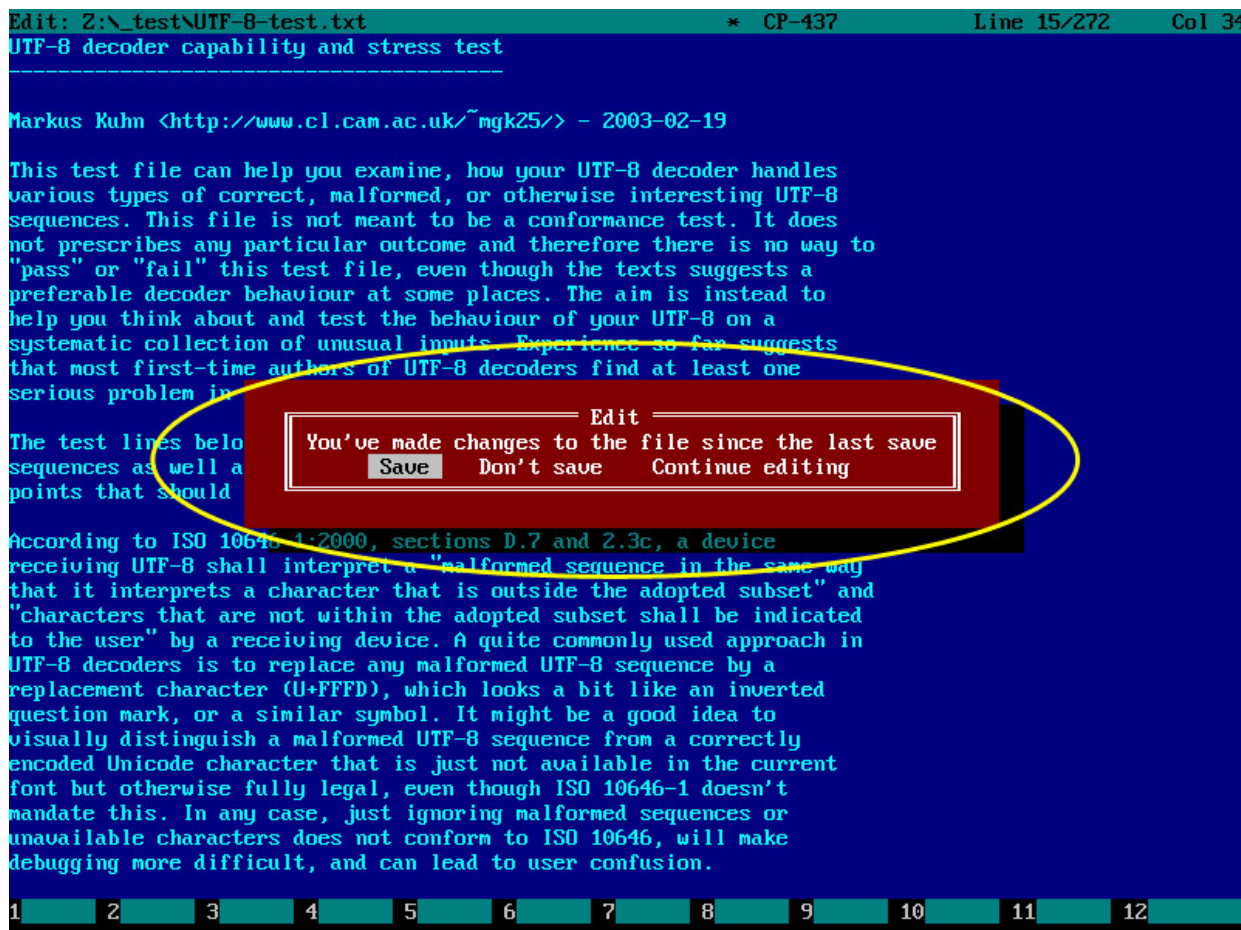
1 help 2 Save 3 4 5 6 7 Search 8 9 10 Quit 11 12 Encoding
```

3. 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 with a blue background and white text. The title bar at the top reads "Edit: Z:_test\UTF-8-test.txt * CP-437 Line 15/272 Col 34". The main text area contains the following content:

```
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 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

The test lines below
sequences as well a
points that should

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.
```

A red dialog box is overlaid on the text, titled "Edit". It contains the message "You've made changes to the file since the last save" and three buttons: "Save", "Don't save", and "Continue editing". A yellow oval highlights the dialog box.

At the bottom of the editor window, there is a status bar with line numbers 1 through 12.

3.2. Save file

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

```
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 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.

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

3.3. Scroll file

To move text cursor within file, use following keys:

Operation	Key
Move cursor to previous line	↑
Move cursor to next line	↓
Move cursor to previous character	←
Move cursor to 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
	Ctrl+Home
Move cursor to the last line in file	Ctrl+PageDown
	Ctrl+End
Scroll view to next page	Page Up
Scroll view to previous page	Page Down

3.4. Input new text or delete text

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

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

To delete text, use following keys:

Operation	Key
Delete character under cursor or merge lines	Delete
Delete character, previous to character under cursor, or merge lines; then move cursor back	Backspace
Delete 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 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.

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

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 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
```

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
-----

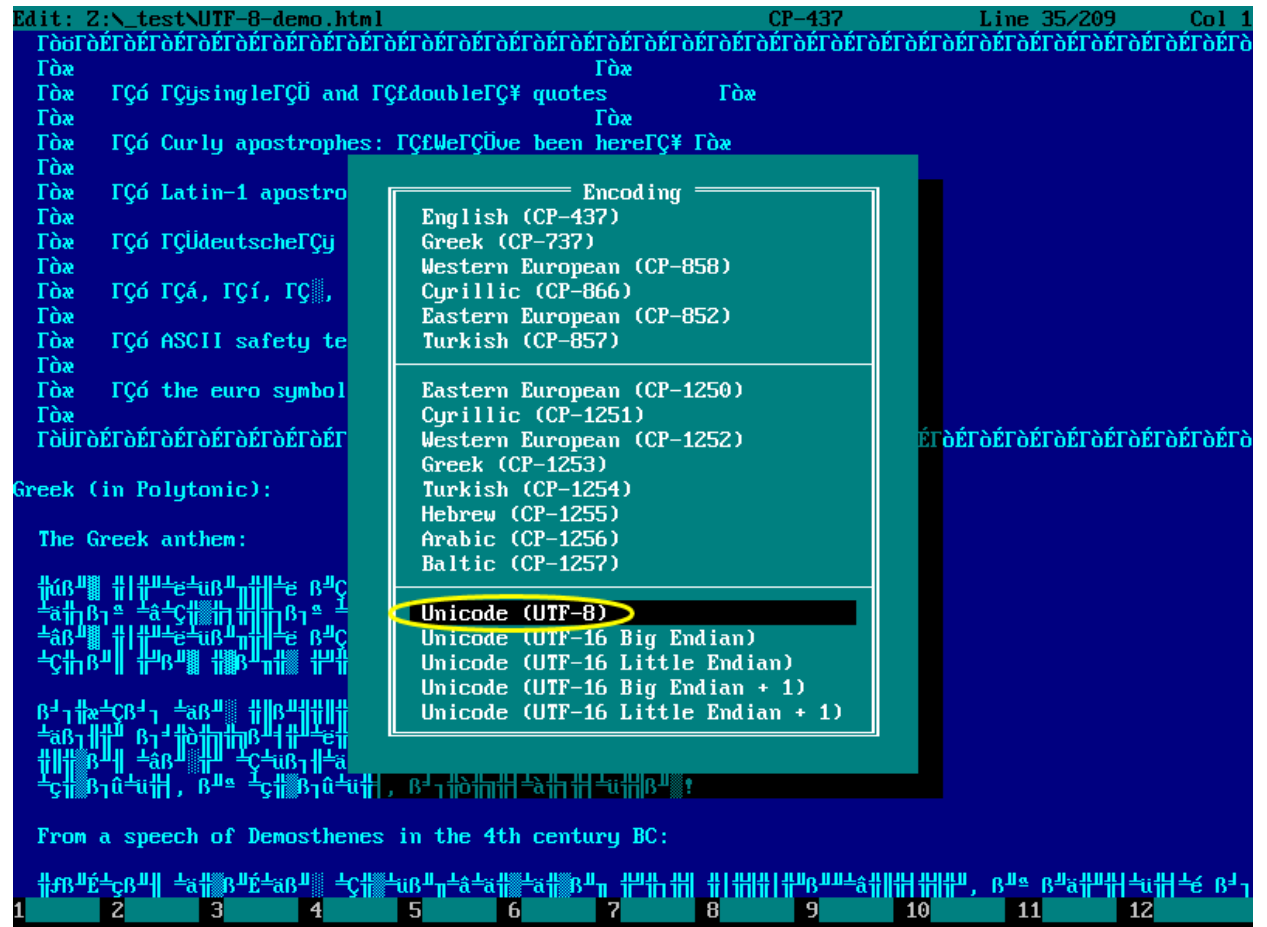
This test file can help you examine, how your UTF-8 decoder handles
various types of correct, malformed, or otherwise interesting UTF-8
sequences. This file is not meant to be a conformance test. It does
not prescribes any particular outcome and therefore there is no way to
"pass" or "fail" this test 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.

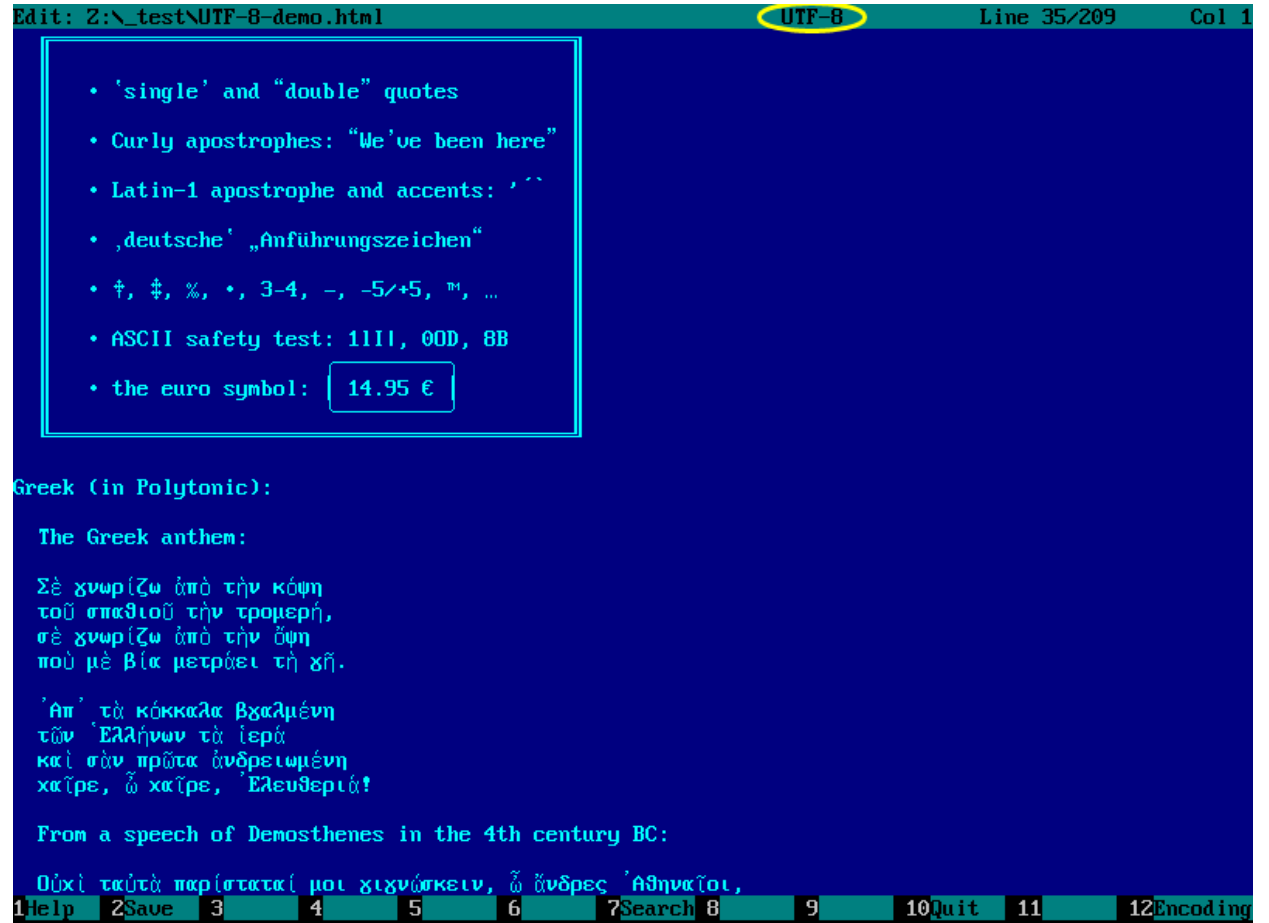
According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving 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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding
```


Choose new encoding in the list within popup window using ↑ and ↓ keys, then **Enter** or mouse click.



Screenshot shows text after encoding changed .



3.7. Delete, copy, move block

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

Block	Non-block
Code page CP437 is based on ASCII, with the following modifications	Code page CP437 is based on ASCII, with the following modifications

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

Select block

To select block, use following keys:

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

Screenshot shows an example of selected block.

```
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.

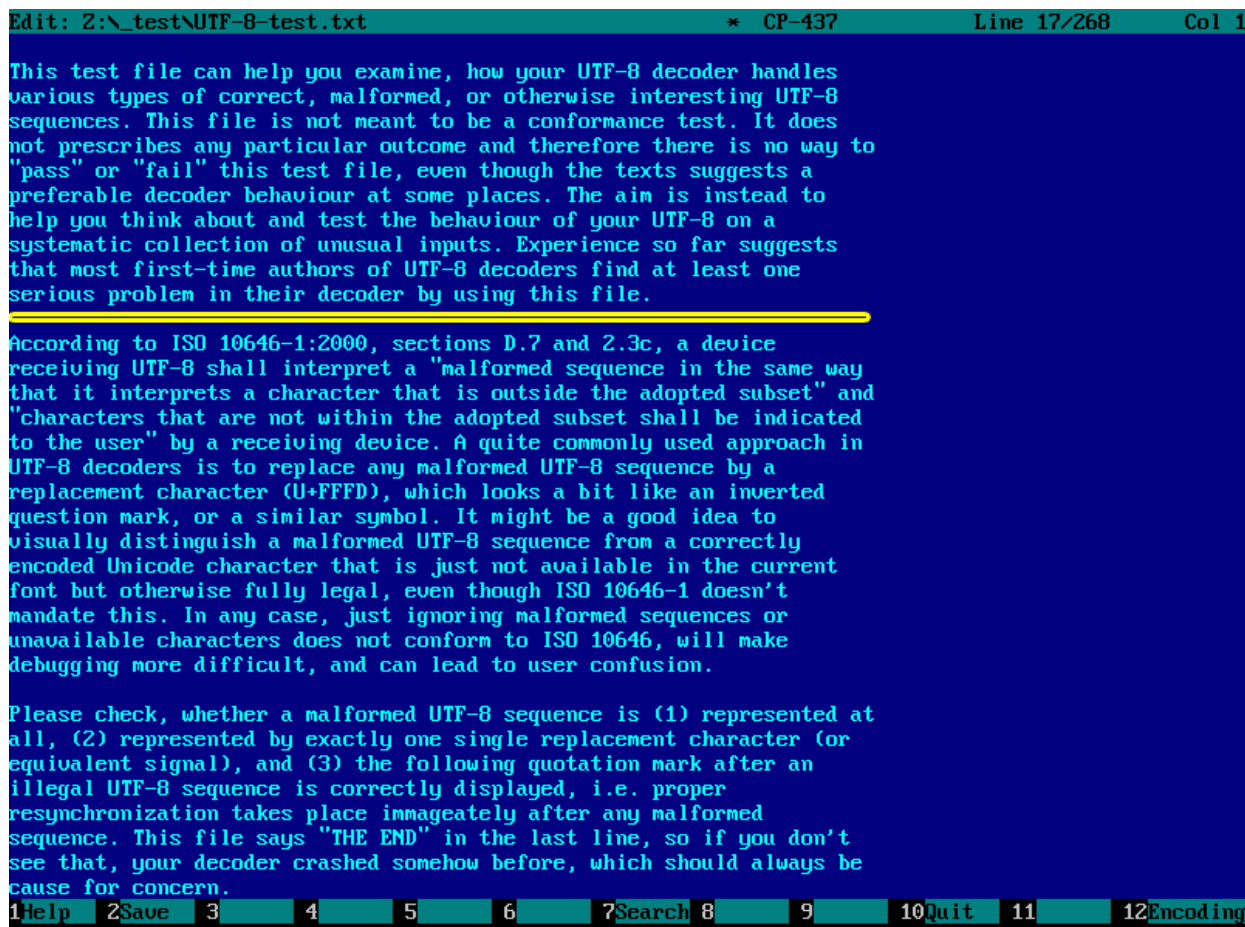
According to ISO 10646-1:2000, sections D.7 and 2.3c, a device
receiving UTF-8 shall interpret a "malformed sequence in the same way
that it interprets a character that is outside the adopted subset" and
"characters that are not within the adopted subset shall be indicated
to the user" by a receiving 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
```

Delete block

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

Screenshot shows location of former block.



The screenshot shows a text editor window with a dark blue background and light blue text. The title bar at the top reads "Edit: Z:_test\UTF-8-test.txt * CP-437 Line 17/268 Col 1". The main text area contains three paragraphs. The first paragraph is highlighted with a yellow background. The second paragraph is underlined with a yellow line. The third paragraph is not highlighted. 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.

```
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.

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

Copy block

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

After block is selected, press **Ctrl+C** or **Ctrl+Insert**, to copy it to 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 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
```

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

Screenshot is taken before insertion of block from 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 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
```

Screenshot is taken after insertion of block from 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 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.

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 block

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

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

Please note that selected block is deleted from the file at once, but its copy is kept in 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 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.
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding
```

Then move cursor to position, which is intended to be a start of inserted block, and then press **Ctrl+V** or **Shift+Insert**.

Screenshot is taken before insertion of block from 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 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
```

Screenshot is taken after insertion of block from 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.

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
```

3.8. Delete, copy, move rectangular block

Rectangular block is a part of text which is either a substring of one string, or set of substrings of multiple adjacent strings, cut in such way that starting and ending positions of all substrings are constants.

Rectangular block	Non-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 delete, copied or moved, it must be selected.

Select rectangular block

To select rectangular block, use following keys:

Operation	Key
Select rectangular block upward from cursor position	Alt+Shift+↑
Select rectangular block downward from cursor position	Alt+Shift+↓
Select rectangular block leftward from cursor position	Alt+Shift+←
Select rectangular block rightward from cursor position	Alt+Shift+→
Select rectangular block from cursor to the start of current line	Alt+Shift+Home
Select rectangular block from cursor to the end of current line	Alt+Shift+End
Select rectangular block from cursor to the start of file	Alt+Ctrl+Shift+Home
Select rectangular block from 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-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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding
```

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-001FFFFF):           "    "
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
1help  2Save  3      4      5      6      7Search 8      9      10Quit 11      12Encoding

```

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

```

Then move cursor to location, 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 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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding

```

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

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 clipboard.

Please note that selected rectangular block is deleted from the file at once, but its copy is kept in 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-001FFFFF):            "    "
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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding

```

Then move cursor to location, 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 clipboard.

Screenshot is taken before insertion of rectangular block from 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): "        "
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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding

```

Screenshot is taken after insertion of rectangular block from clipboard.

```
Edit: Z:\_testUTF-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): " " 1 byte
2.2.2 (U-000007FF): "  " 2 bytes
2.2.3 (U-0000FFFF): "   " 3 bytes
2.2.4 (U-001FFFFFF): "    " 4 bytes
2.2.5 (U-03FFFFFF): "     " 5 bytes
2.2.6 (U-7FFFFFFF): "      " 6 bytes
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
1help 2Save 3 4 5 6 7Search 8 9 10Quit 11 12Encoding
```