VIP

A Vi Package for GNU Emacs
(Version 3.5, September 15, 1987)

Masahiko Sato

Distribution

Introduction

VIP is a Vi emulating package written in Emacs Lisp. VIP implements most Vi commands including Ex commands. It is therefore hoped that this package will enable you to do Vi style editing under the powerful GNU Emacs environment. This manual describes the usage of VIP assuming that you are fairly accustomed to Vi but not so much with Emacs. Also we will concentrate mainly on differences from Vi, especially features unique to VIP.

It is recommended that you read chapters on survey and on customization before you start using VIP. Other chapters may be used as future references.

Comments and bug reports are welcome. Please send messages to `ms@Sail.Stanford.Edu` if you are outside of Japan and to `masahiko@unsun.riec.tohoku.junet` if you are in Japan.

1 A Survey of VIP

In this chapter we describe basics of VIP with emphasis on the features not found in Vi and on how to use VIP under GNU Emacs.

1.1 Basic Concepts

We begin by explaining some basic concepts of Emacs. These concepts are explained in more detail in the GNU Emacs Manual.

Conceptually, a *buffer* is just a string of ASCII characters and two special characters ⟨PNT⟩ (*point*) and ⟨MRK⟩ (*mark*) such that the character ⟨PNT⟩ occurs exactly once and ⟨MRK⟩ occurs at most once. The *text* of a buffer is obtained by deleting the occurrences of ⟨PNT⟩ and ⟨MRK⟩. If, in a buffer, there is a character following ⟨PNT⟩ then we say that point is *looking at* the character; otherwise we say that point is *at the end of buffer*. ⟨PNT⟩ and ⟨MRK⟩ are used to indicate positions in a buffer and they are not part of the text of the buffer. If a buffer contains a ⟨MRK⟩ then the text between ⟨MRK⟩ and ⟨PNT⟩ is called the *region* of the buffer.

Emacs provides (multiple) *windows* on the screen, and you can see the content of a buffer through the window associated with the buffer. The cursor of the screen is always positioned on the character after ⟨PNT⟩.

A *keymap* is a table that records the bindings between characters and command functions. There is the *global keymap* common to all the buffers. Each buffer has its *local keymap* that determines the *mode* of the buffer. Local keymap overrides global keymap, so that if a function is bound to some key in the local keymap then that function will be executed when you type the key. If no function is bound to a key in the local map, however, the function bound to the key in the global map becomes in effect.

1.2 Loading VIP

The recommended way to load VIP automatically is to include the line:

```
(load "vip")
```

in your '`.emacs`' file. The '`.emacs`' file is placed in your home directory and it will be executed every time you invoke Emacs. If you wish to be in vi mode whenever Emacs starts up, you can include the following line in your '`.emacs`' file instead of the above line:

```
(setq term-setup-hook 'vip-mode)
```

(See Section 1.3.2 [Vi Mode], page 4, for the explanation of vi mode.)

Even if your '`.emacs`' file does not contain any of the above lines, you can load VIP and enter vi mode by typing the following from within Emacs.

```
M-x vip-mode
```

1.3 Modes in VIP

Loading VIP has the effect of globally binding *C-z* (*Control-z*) to the function `vip-change-mode-to-vi`. The default binding of *C-z* in GNU Emacs is `suspend-emacs`, but, you can also call `suspend-emacs` by typing *C-x C-z*. Other than this, all the key bindings of Emacs remain the same after loading VIP.

Now, if you hit *C-z*, the function `vip-change-mode-to-vi` will be called and you will be in *vi mode*. (Some major modes may locally bind *C-z* to some special functions. In such cases, you can call `vip-change-mode-to-vi` by `execute-extended-command` which is invoked by *M-x*. Here *M-x* means *Meta-x*, and if your terminal does not have a ⟨META⟩ key you can enter it by typing ⟨ESC⟩ *x*. The same effect can also be achieve by typing *M-x vip-mode*.)

You can observe the change of mode by looking at the *mode line*. For instance, if the mode line is:

```
-----Emacs: *scratch*              (Lisp Interaction)----All------------
```

then it will change to:

```
-----Vi:    *scratch*              (Lisp Interaction)----All------------
```

Thus the word '`Emacs`' in the mode line will change to '`Vi`'.

You can go back to the original *emacs mode* by typing *C-z* in vi mode. Thus *C-z* toggles between these two modes.

Note that modes in VIP exist orthogonally to modes in Emacs. This means that you can be in vi mode and at the same time, say, shell mode.

Vi mode corresponds to Vi's command mode. From vi mode you can enter *insert mode* (which corresponds to Vi's insert mode) by usual Vi command keys like *i*, `a`, *o* . . . etc.

In insert mode, the mode line will look like this:

```
-----Insert *scratch*              (Lisp Interaction)----All------------
```

You can exit from insert mode by hitting ⟨ESC⟩ key as you do in Vi.

That VIP has three modes may seem very complicated, but in fact it is not so. VIP is implemented so that you can do most editing remaining only in the two modes for Vi (that is vi mode and insert mode).

1.3.1 Emacs Mode

You will be in this mode just after you loaded VIP. You can do all normal Emacs editing in this mode. Note that the key *C-z* is globally bound to `vip-change-mode-to-vi`. So, if you type *C-z* in this mode then you will be in vi mode.

1.3.2 Vi Mode

This mode corresponds to Vi's command mode. Most Vi commands work as they do in Vi. You can go back to emacs mode by typing *C-z*. You can enter insert mode, just as in Vi, by typing *i*, `a` etc.

1.3.3 Insert Mode

The key bindings in this mode is the same as in the emacs mode except for the following 4 keys. So, you can move around in the buffer and change its content while you are in insert mode.

⟨ESC⟩ This key will take you back to vi mode.

C-h Delete previous character.

C-w Delete previous word.

C-z Typing this key has the same effect as typing ⟨ESC⟩ in emacs mode. Thus typing *C-z x* in insert mode will have the same effect as typing *ESC x* in emacs mode.

1.4 Differences from Vi

The major differences from Vi are explained below.

1.4.1 Undoing

You can repeat undoing by the `.` key. So, *u* will undo a single change, while *u . . .*, for instance, will undo 4 previous changes. Undo is undoable as in Vi. So the content of the buffer will be the same before and after *u u*.

1.4.2 Changing

Some commands which change a small number of characters are executed slightly differently. Thus, if point is at the beginning of a word '`foo`' and you wished to change it to '`bar`' by typing *c w*, then VIP will prompt you for a new word in the minibuffer by the prompt '`foo =>` '. You can then enter '`bar`' followed by ⟨RET⟩ or ⟨ESC⟩ to complete the command. Before you enter ⟨RET⟩ or ⟨ESC⟩ you can abort the command by typing *C-g*. In general, you can abort a partially formed command by typing *C-g*.

1.4.3 Searching

As in Vi, searching is done by */* and *?*. The string will be searched literally by default. To invoke a regular expression search, first execute the search command */* (or *?*) with empty search string. (I.e, type */* followed by ⟨RET⟩.) A search for empty string will toggle the search mode between vanilla search and regular expression search. You cannot give an offset to the search string. (It is a limitation.) By default, search will wrap around the buffer as in Vi. You can change this by rebinding the variable `vip-search-wrap-around`. See Chapter 4 [Customization], page 26, for how to do this.

1.4.4 z Command

For those of you who cannot remember which of *z* followed by ⟨RET⟩, `.` and `-` do what. You can also use *z* followed by *H*, *M* and *L* to place the current line in the Home (Middle, and Last) line of the window.

1.4.5 Counts

Some Vi commands which do not accept a count now accept one

`p`
`P` Given counts, text will be yanked (in Vi's sense) that many times. Thus `3 p` is the same as `p p p`.

`o`
`O` Given counts, that many copies of text will be inserted. Thus `o a b c` ⟨ESC⟩ will insert 3 lines of 'abc' below the current line.

`/`
`?` Given a count n, n-th occurrence will be searched.

1.4.6 Marking

Typing an `m` followed by a lower-case character *ch* marks the point to the register named *ch* as in Vi. In addition to these, we have following key bindings for marking.

`m <` Set mark at the beginning of buffer.

`m >` Set mark at the end of buffer.

`m .` Set mark at point (and push old mark on mark ring).

`m ,` Jump to mark (and pop mark off the mark ring).

1.4.7 Region Commands

Vi operators like `d`, `c` etc. are usually used in combination with motion commands. It is now possible to use current region as the argument to these operators. (A *region* is a part of buffer delimited by point and mark.) The key `r` is used for this purpose. Thus `d r` will delete the current region. If `R` is used instead of `r` the region will first be enlarged so that it will become the smallest region containing the original region and consisting of whole lines. Thus `m . d R` will have the same effect as `d d`.

1.4.8 Some New Commands

Note that the keys below (except for `R`) are not used in Vi.

`C-a` Move point to the beginning of line.

`C-n` If you have two or more windows in the screen, this key will move point to the next window.

`C-o` Insert a newline and leave point before it, and then enter insert mode.

`C-r` Backward incremental search.

`C-s` Forward incremental search.

`C-c`
`C-x`
⟨ESC⟩ These keys will exit from vi mode and return to emacs mode temporarily. If you hit one of these keys, Emacs will be in emacs mode and will believe that

you hit that key in emacs mode. For example, if you hit *C-x* followed by *2*, then the current window will be split into 2 and you will be in vi mode again.

`\` Escape to emacs mode. Hitting `\` will take you to emacs mode, and you can execute a single Emacs command. After executing the Emacs command you will be in vi mode again. You can give a count before typing `\`. Thus *5 \ **, as well as *\ C-u 5 **, will insert '`*****`' before point. Similarly *1 0 \ C-p* will move the point 10 lines above the current line.

K Kill current buffer if it is not modified. Useful when you selected a buffer which you did not want.

Q
R *Q* is for query replace and *R* is for replace. By default, string to be replaced are treated literally. If you wish to do a regular expression replace, first do replace with empty string as the string to be replaced. In this way, you can toggle between vanilla and regular expression replacement.

v
V These keys are used to Visit files. *v* will switch to a buffer visiting file whose name can be entered in the minibuffer. *V* is similar, but will use window different from the current window.

If followed by a certain character *ch*, it becomes an operator whose argument is the region determined by the motion command that follows. Currently, *ch* can be one of *c*, *C*, *g*, *q* and *s*.

 # c Change upper-case characters in the region to lower case (`downcase-region`).

 # C Change lower-case characters in the region to upper case. For instance, *# C 3 w* will capitalize 3 words from the current point (`upcase-region`).

 # g Execute last keyboard macro for each line in the region (`vip-global-execute`).

 # q Insert specified string at the beginning of each line in the region (`vip-quote-region`).

 # s Check spelling of words in the region (`spell-region`).

*** Call last keyboard macro.

1.4.9 New Key Bindings

In VIP the meanings of some keys are entirely different from Vi. These key bindings are done deliberately in the hope that editing under Emacs will become easier. It is however possible to rebind these keys to functions which behave similarly as in Vi. See Section 4.2 [Customizing Key Bindings], page 26, for details.

C-g
g In Vi, *C-g* is used to get information about the file associated to the current buffer. Here, *g* will do that, and *C-g* is used to abort a command (this is for compatibility with emacs mode.)

SPC
⟨RET⟩ Now these keys will scroll up and down the text of current window. Convenient for viewing the text.

s
S They are used to switch to a specified buffer. Useful for switching to already existing buffer since buffer name completion is provided. Also a default buffer will be given as part of the prompt, to which you can switch by just typing ⟨RET⟩ key. *s* is used to select buffer in the current window, while *S* selects buffer in another window.

C
X These keys will exit from vi mode and return to emacs mode temporarily. If you type *C* (*X*), Emacs will be in emacs mode and will believe that you have typed *C-c* (*C-x*, resp.) in emacs mode. Moreover, if the following character you type is an upper-case letter, then Emacs will believe that you have typed the corresponding control character. You will be in vi mode again after the command is executed. For example, typing *X S* in vi mode is the same as typing *C-x C-s* in emacs mode. You get the same effect by typing *C-x C-s* in vi mode, but the idea here is that you can execute useful Emacs commands without typing control characters. For example, if you hit *X* (or *C-x*) followed by *2*, then the current window will be split into 2 and you will be in vi mode again.

In addition to these, `ctl-x-map` is slightly modified:

X 3
C-x 3 This is equivalent to *C-x 1 C-x 2* (1 + 2 = 3).

1.4.10 Window Commands

In this and following subsections, we give a summary of key bindings for basic functions related to windows, buffers and files.

C-n Switch to next window.

X 1
C-x 1 Delete other windows.

X 2
C-x 2 Split current window into two windows.

X 3
C-x 3 Show current buffer in two windows.

1.4.11 Buffer Commands

s Switch to the specified buffer in the current window (`vip-switch-to-buffer`).

S Switch to the specified buffer in another window (`vip-switch-to-buffer-other-window`).

K Kill the current buffer if it is not modified.

X S
C-x C-s Save the current buffer in the file associated to the buffer.

1.4.12 File Commands

v Visit specified file in the current window.

V Visit specified file in another window.

X W
C-x C-w Write current buffer into the specified file.

X I
C-x C-i
Insert specified file at point.

1.4.13 Miscellaneous Commands

X (
C-x (Start remembering keyboard macro.

X)
C-x) Finish remembering keyboard macro.

*** Call last remembered keyboard macro.

X Z
C-x C-z Suspend Emacs.

Z Z Exit Emacs.
Q Query replace.
R Replace.

2 Vi Commands

This chapter describes Vi commands other than Ex commands implemented in VIP. Except for the last section which discusses insert mode, all the commands described in this chapter are to be used in vi mode.

2.1 Numeric Arguments

Most Vi commands accept a *numeric argument* which can be supplied as a prefix to the commands. A numeric argument is also called a *count*. In many cases, if a count is given, the command is executed that many times. For instance, `5 d d` deletes 5 lines while simple `d d` deletes a line. In this manual the metavariable *n* will denote a count.

2.2 Important Keys

The keys `C-g` and `C-l` are unique in that their associated functions are the same in any of emacs, vi and insert mode.

`C-g` Quit. Cancel running or partially typed command (`keyboard-quit`).

`C-l` Clear the screen and reprint everything (`recenter`).

In Emacs many commands are bound to the key strokes that start with `C-x`, `C-c` and ⟨ESC⟩. These commands can be accessed from vi mode as easily as from emacs mode.

`C-x`
`C-c`
⟨ESC⟩ Typing one of these keys have the same effect as typing it in emacs mode. Appropriate command will be executed according as the keys you type after it. You will be in vi mode again after the execution of the command. For instance, if you type ⟨ESC⟩ `<` (in vi mode) then the cursor will move to the beginning of the buffer and you will still be in vi mode.

`C`
`X` Typing one of these keys have the effect of typing the corresponding control character in emacs mode. Moreover, if you type an upper-case character following it, that character will also be translated to the corresponding control character. Thus typing `X W` in vi mode is the same as typing `C-x C-w` in emacs mode. You will be in vi mode again after the execution of a command.

`\` Escape to emacs mode. Hitting the `\` key will take you to emacs mode, and you can execute a single Emacs command. After executing the Emacs command you will be in vi mode again. You can give a count before typing `\`. Thus `5 \ +`, as well as `\ C-u 5 +`, will insert '`+++++`' before point.

2.3 Buffers and Windows

In Emacs the text you edit is stored in a *buffer*. See GNU Emacs Manual, for details. There is always one *selected* buffer which is called the *current buffer*.

You can see the contents of buffers through *windows* created by Emacs. When you have multiple windows on the screen only one of them is selected. Each buffer has a unique name, and each window has a mode line which shows the name of the buffer associated with the window and other information about the status of the buffer. You can change the format of the mode line, but normally if you see '`**`' at the beginning of a mode line it means that the buffer is *modified*. If you write out the content of the buffer to a file, then the buffer will become not modified. Also if you see '`%%`' at the beginning of the mode line, it means that the file associated with the buffer is write protected.

We have the following commands related to windows and buffers.

C-n Move cursor to the next-window (`vip-next-window`).

X 1 Delete other windows and make the selected window fill the screen (`delete-other-windows`).

X 2 Split current window into two windows (`split-window-vertically`).

X 3 Show current buffer in two windows.

s buffer ⟨RET⟩
Select or create a buffer named *buffer* (`vip-switch-to-buffer`).

S buffer ⟨RET⟩
Similar but select a buffer named *buffer* in another window (`vip-switch-to-buffer-other-window`).

K Kill the current buffer if it is not modified or if it is not associated with a file (`vip-kill-buffer`).

X B List the existing buffers (`list-buffers`).

As *buffer name completion* is provided, you have only to type in initial substring of the buffer name which is sufficient to identify it among names of existing buffers. After that, if you hit ⟨TAB⟩ the rest of the buffer name will be supplied by the system, and you can confirm it by ⟨RET⟩. The default buffer name to switch to will also be prompted, and you can select it by giving a simple ⟨RET⟩. See GNU Emacs Manual for details of completion.

2.4 Files

We have the following commands related to files. They are used to visit, save and insert files.

v file ⟨RET⟩
Visit specified file in the current window (`vip-find-file`).

V file ⟨RET⟩
Visit specified file in another window (`vip-find-file-other-window`).

X S Save current buffer to the file associated with the buffer. If no file is associated with the buffer, the name of the file to write out the content of the buffer will be asked in the minibuffer.

X W file ⟨RET⟩
Write current buffer into a specified file.

X I file ⟨RET⟩
: Insert a specified file at point.

g
: Give information on the file associated with the current buffer. Tell you the name of the file associated with the buffer, the line number of the current point and total line numbers in the buffer. If no file is associated with the buffer, this fact will be indicated by the null file name '`""`'.

In Emacs, you can edit a file by *visiting* it. If you wish to visit a file in the current window, you can just type *v*. Emacs maintains the *default directory* which is specific to each buffer. Suppose, for instance, that the default directory of the current buffer is '`/usr/masahiko/lisp/`'. Then you will get the following prompt in the minibuffer.

```
visit file: /usr/masahiko/lisp/
```

If you wish to visit, say, '`vip.el`' in this directory, then you can just type '`vip.el`' followed by ⟨RET⟩. If the file '`vip.el`' already exists in the directory, Emacs will visit that file, and if not, the file will be created. Emacs will use the file name ('`vip.el`', in this case) as the name of the buffer visiting the file. In order to make the buffer name unique, Emacs may append '`<2>`', '`<3>`' etc., to the buffer name. As the *file name completion* is provided here, you can sometime save typing. For instance, suppose there is only one file in the default directory whose name starts with '`v`', that is '`vip.el`'. Then if you just type *v* ⟨TAB⟩ then it will be completed to '`vip.el`'. Thus, in this case, you just have to type *v v* ⟨TAB⟩ ⟨RET⟩ to visit '`/usr/masahiko/lisp/vip.el`'. Continuing the example, let us now suppose that you wished to visit the file '`/usr/masahiko/man/vip.texinfo`'. Then to the same prompt which you get after you typed *v*, you can enter '`/usr/masahiko/man/vip.texinfo`' or '`../man/vip.texinfo`' followed by ⟨RET⟩.

Use *V* instead of *v*, if you wish to visit a file in another window.

You can verify which file you are editing by typing *g*. (You can also type *X B* to get nformation on other buffers too.) If you type *g* you will get an information like below in the echo area:

```
"/usr/masahiko/man/vip.texinfo" line 921 of 1949
```

After you edited the buffer ('`vip.texinfo`', in our example) for a while, you may wish to save it in a file. If you wish to save it in the file associated with the buffer ('`/usr/masahiko/man/vip.texinfo`', in this case), you can just say *X S*. If you wish to save it in another file, you can type *X W*. You will then get a similar prompt as you get for *v*, to which you can enter the file name.

2.5 Viewing the Buffer

In this and next section we discuss commands for moving around in the buffer. These command do not change the content of the buffer. The following commands are useful for viewing the content of the current buffer.

⟨SPC⟩
C-f
: Scroll text of current window upward almost full screen. You can go *forward* in the buffer by this command (`vip-scroll`).

⟨RET⟩
C-b Scroll text of current window downward almost full screen. You can go *backward* in the buffer by this command (`vip-scroll-back`).

C-d Scroll text of current window upward half screen. You can go *down* in the buffer by this command (`vip-scroll-down`).

C-u Scroll text of current window downward half screen. You can go *up* in the buffer by this command (`vip-scroll-up`).

C-y Scroll text of current window upward by one line (`vip-scroll-down-one`).

C-e Scroll text of current window downward by one line (`vip-scroll-up-one`).

You can repeat these commands by giving a count. Thus, *2* ⟨SPC⟩ has the same effect as ⟨SPC⟩ ⟨SPC⟩.

The following commands reposition point in the window.

z H
z ⟨RET⟩ Put point on the top (*home*) line in the window. So the current line becomes the top line in the window. Given a count n, point will be placed in the n-th line from top (`vip-line-to-top`).

z M
z . Put point on the *middle* line in the window. Given a count n, point will be placed in the n-th line from the middle line (`vip-line-to-middle`).

z L
z - Put point on the *bottom* line in the window. Given a count n, point will be placed in the n-th line from bottom (`vip-line-to-bottom`).

C-l Center point in window and redisplay screen (`recenter`).

2.6 Mark Commands

The following commands are used to mark positions in the buffer.

m ch Store current point in the register *ch*. *ch* must be a lower-case ASCII letter.

m < Set mark at the beginning of current buffer.

m > Set mark at the end of current buffer.

m . Set mark at point.

m , Jump to mark (and pop mark off the mark ring).

Emacs uses the *mark ring* to store marked positions. The commands *m <*, *m >* and *m .* not only set mark but also add it as the latest element of the mark ring (replacing the oldest one). By repeating the command '*m ,*' you can visit older and older marked positions. You will eventually be in a loop as the mark ring is a ring.

2.7 Motion Commands

Commands for moving around in the current buffer are collected here. These commands are used as an 'argument' for the delete, change and yank commands to be described in the next section.

`h` Move point backward by one character. Signal error if point is at the beginning of buffer, but (unlike Vi) do not complain otherwise (`vip-backward-char`).

`l` Move point backward by one character. Signal error if point is at the end of buffer, but (unlike Vi) do not complain otherwise (`vip-forward-char`).

`j` Move point to the next line keeping the current column. If point is on the last line of the buffer, a new line will be created and point will move to that line (`vip-next-line`).

`k` Move point to the previous line keeping the current column (`vip-next-line`).

`+` Move point to the next line at the first non-white character. If point is on the last line of the buffer, a new line will be created and point will move to the beginning of that line (`vip-next-line-at-bol`).

`-` Move point to the previous line at the first non-white character (`vip-previous-line-at-bol`).

If a count is given to these commands, the commands will be repeated that many times.

`0` Move point to the beginning of line (`vip-beginning-of-line`).

`^` Move point to the first non-white character on the line (`vip-bol-and-skip-white`).

`$` Move point to the end of line (`vip-goto-eol`).

n `|` Move point to the *n*-th column on the line (`vip-goto-col`).

Except for the `|` command, these commands neglect a count.

`w` Move point forward to the beginning of the next word (`vip-forward-word`).

`W` Move point forward to the beginning of the next word, where a *word* is considered as a sequence of non-white characters (`vip-forward-Word`).

`b` Move point backward to the beginning of a word (`vip-backward-word`).

`B` Move point backward to the beginning of a word, where a *word* is considered as a sequence of non-white characters (`vip-forward-Word`).

`e` Move point forward to the end of a word (`vip-end-of-word`).

`E` Move point forward to the end of a word, where a *word* is considered as a sequence of non-white characters (`vip-end-of-Word`).

Here the meaning of the word 'word' for the `w`, `b` and `e` commands is determined by the *syntax table* effective in the current buffer. Each major mode has its syntax mode, and therefore the meaning of a word also changes as the major mode changes. See GNU Emacs Manual for details of syntax table.

H Move point to the beginning of the *home* (top) line of the window. Given a count n, go to the n-th line from top (`vip-window-top`).

M Move point to the beginning of the *middle* line of the window. Given a count n, go to the n-th line from the middle line (`vip-window-middle`).

L Move point to the beginning of the *lowest* (bottom) line of the window. Given count, go to the n-th line from bottom (`vip-window-bottom`).

These commands can be used to go to the desired line visible on the screen.

(Move point backward to the beginning of the sentence (`vip-backward-sentence`).

) Move point forward to the end of the sentence (`vip-forward-sentence`).

{ Move point backward to the beginning of the paragraph (`vip-backward-paragraph`).

} Move point forward to the end of the paragraph (`vip-forward-paragraph`).

A count repeats the effect for these commands.

G Given a count n, move point to the n-th line in the buffer on the first non-white character. Without a count, go to the end of the buffer (`vip-goto-line`).

‘ ‘ Exchange point and mark (`vip-goto-mark`).

‘ ch Move point to the position stored in the register *ch*. *ch* must be a lower-case letter.

’ ’ Exchange point and mark, and then move point to the first non-white character on the line (`vip-goto-mark-and-skip-white`).

’ ch Move point to the position stored in the register *ch* and skip to the first non-white character on the line. *ch* must be a lower-case letter.

% Move point to the matching parenthesis if point is looking at *(*, *)*, *{*, *}*, *[* or *]* (`vip-paren-match`).

The command *G* mark point before move, so that you can return to the original point by *‘ ‘*. The original point will also be stored in the mark ring.

The following commands are useful for moving points on the line. A count will repeat the effect.

f ch Move point forward to the character *ch* on the line. Signal error if *ch* could not be found (`vip-find-char-forward`).

F ch Move point backward to the character *ch* on the line. Signal error if *ch* could not be found (`vip-find-char-backward`).

t ch Move point forward upto the character *ch* on the line. Signal error if *ch* could not be found (`vip-goto-char-forward`).

T ch Move point backward upto the character *ch* on the line. Signal error if *ch* could not be found (`vip-goto-char-backward`).

; Repeat previous *f*, *t*, *F* or *T* command (`vip-repeat-find`).

, Repeat previous *f*, *t*, *F* or *T* command, in the opposite direction (`vip-repeat-find-opposite`).

2.8 Searching and Replacing

Following commands are available for searching and replacing.

/ string ⟨RET⟩
: Search the first occurrence of the string *string* forward starting from point. Given a count *n*, the *n*-th occurrence of *string* will be searched. If the variable `vip-re-search` has value `t` then *regular expression* search is done and the string matching the regular expression *string* is found. If you give an empty string as *string* then the search mode will change from vanilla search to regular expression search and vice versa (`vip-search-forward`).

? string ⟨RET⟩
: Same as `/`, except that search is done backward (`vip-search-backward`).

n
: Search the previous search pattern in the same direction as before (`vip-search-next`).

N
: Search the previous search pattern in the opposite direction (`vip-search-Next`).

C-s
: Search forward incrementally. See GNU Emacs Manual for details (`isearch-forward`).

C-r
: Search backward incrementally (`isearch-backward`).

R string RET newstring
: There are two modes of replacement, *vanilla* and *regular expression*. If the mode is *vanilla* you will get a prompt '`Replace string:`', and if the mode is *regular expression* you will ge a prompt '`Replace regexp:`'. The mode is initially *vanilla*, but you can toggle these modes by giving a null string as *string*. If the mode is vanilla, this command replaces every occurrence of *string* with *newstring*. If the mode is regular expression, *string* is treated as a regular expression and every string matching the regular expression is replaced with *newstring* (`vip-replace-string`).

Q string RET newstring
: Same as *R* except that you will be asked form confirmation before each replacement
(`vip-query-replace`).

r ch
: Replace the character point is looking at by the character *ch*. Give count, replace that many characters by *ch* (`vip-replace-char`).

The commands `/` and `?` mark point before move, so that you can return to the original point by *` `*.

2.9 Modifying Commands

In this section, commands for modifying the content of a buffer are described. These commands affect the region determined by a motion command which is given to the commands as their argument.

We classify motion commands into *point commands* and *line commands*. The point commands are as follows:

`h, l, 0, ^, $, w, W, b, B, e, E, (, ), /, ?, ‘, f, F, t, T, %, ;, ,`

The line commands are as follows:

`j, k, +, -, H, M, L, {, }, G, ’`

If a point command is given as an argument to a modifying command, the region determined by the point command will be affected by the modifying command. On the other hand, if a line command is given as an argument to a modifying command, the region determined by the line command will be enlarged so that it will become the smallest region properly containing the region and consisting of whole lines (we call this process *expanding the region*), and then the enlarged region will be affected by the modifying command.

2.9.1 Delete Commands

`d` *motion-command*
: Delete the region determined by the motion command *motion-command*.

For example, `d $` will delete the region between point and end of current line since `$` is a point command that moves point to end of line. `d G` will delete the region between the beginning of current line and end of the buffer, since `G` is a line command. A count given to the command above will become the count for the associated motion command. Thus, `3 d w` will delete three words.

It is also possible to save the deleted text into a register you specify. For example, you can say `" t 3 d w` to delete three words and save it to register `t`. The name of a register is a lower-case letter between `a` and `z`. If you give an upper-case letter as an argument to a delete command, then the deleted text will be appended to the content of the register having the corresponding lower-case letter as its name. So, `" T d w` will delete a word and append it to register `t`. Other modifying commands also accept a register name as their argument, and we will not repeat similar explanations.

We have more delete commands as below.

`d d` Delete a line. Given a count n, delete n lines.

`d r` Delete current region.

`d R` Expand current region and delete it.

`D` Delete to the end of a line (`vip-kill-line`).

`x` Delete a character after point. Given n, delete n characters (`vip-delete-char`).

⟨DEL⟩ Delete a character before point. Given n, delete n characters (`vip-delete-backward-char`).

2.9.2 Yank Commands

Yank commands *yank* a text of buffer into a (usually anonymous) register. Here the word ‘yank’ is used in Vi’s sense. Thus yank commands do not alter the content of the buffer, and useful only in combination with commands that put back the yanked text into the buffer.

y motion-command
: Yank the region determined by the motion command *motion-command*.

For example, *y $* will yank the text between point and the end of line into an anonymous register, while *"c y $* will yank the same text into register *c*.

Use the following command to yank consecutive lines of text.

y y
Y
: Yank a line. Given *n*, yank *n* lines (`vip-yank-line`).

y r
: Yank current region.

y R
: Expand current region and yank it.

2.9.3 Put Back Commands

Deleted or yanked texts can be put back into the buffer by the command below.

p
: Insert, after the character point is looking at, most recently deleted/yanked text from anonymous register. Given a register name argument, the content of the named register will be put back. Given a count, the command will be repeated that many times. This command also checks if the text to put back ends with a new line character, and if so the text will be put below the current line (`vip-put-back`).

P
: Insert at point most recently deleted/yanked text from anonymous register. Given a register name argument, the content of the named register will be put back. Given a count, the command will be repeated that many times. This command also checks if the text to put back ends with a new line character, and if so the text will be put above the current line rather than at point (`vip-Put-back`).

Thus, *" c p* will put back the content of the register *c* into the buffer. It is also possible to specify *number register* which is a numeral between *1* and *9*. If the number register *n* is specified, *n*-th previously deleted/yanked text will be put back. It is an error to specify a number register for the delete/yank commands.

2.9.4 Change Commands

Most commonly used change command takes the following form.

c motion-command
: Replace the content of the region determined by the motion command *motion-command* by the text you type. If the motion command is a point command then you will type the text into minibuffer, and if the motion command is a line command then the region will be deleted first and you can insert the text in *insert mode*.

For example, if point is at the beginning of a word '`foo`' and you wish to change it to '`bar`', you can type *c w*. Then, as *w* is a point command, you will get the prompt '`foo =>`' in the minibuffer, for which you can type *b a r* ⟨RET⟩ to complete the change command.

c c
: Change a line. Given a count, that many lines are changed.

c r Change current region.

c R Expand current region and change it.

2.9.5 Repeating and Undoing Modifications

VIP records the previous modifying command, so that it is easy to repeat it. It is also very easy to undo changes made by modifying commands.

u Undo the last change. You can undo more by repeating undo by the repeat command '`.`'. For example, you can undo 5 previous changes by typing '`u....`'. If you type '`uu`', then the second '`u`' undoes the first undo command (`vip-undo`).

. Repeat the last modifying command. Given count *n* it becomes the new count for the repeated command. Otherwise, the count for the last modifying command is used again (`vip-repeat`).

2.10 Other Vi Commands

Miscellaneous Vi commands are collected here.

Z Z Exit Emacs. If modified buffers exist, you will be asked whether you wish to save them or not (`save-buffers-kill-emacs`).

! motion-command format-command
n ! ! format-command
The region determined by the motion command *motion-command* will be given to the shell command *format-command* and the region will be replaced by its output. If a count is given, it will be passed to *motion-command*. For example, '`3!Gsort`' will sort the region between point and the 3rd line. If *!* is used instead of *motion-command* then *n* lines will be processed by *format-command* (`vip-command-argument`).

J Join two lines. Given count, join that many lines. A space will be inserted at each junction (`vip-join-lines`).

< motion-command
n < < Shift region determined by the motion command *motion-command* to left by *shift-width* (default is 8). If `<` is used instead of *motion-command* then shift *n* lines
(`vip-command-argument`).

> motion-command
n > > Shift region determined by the motion command *motion-command* to right by *shift-width* (default is 8). If `<` is used instead of *motion-command* then shift *n* lines
(`vip-command-argument`).

= motion-command
Indent region determined by the motion command *motion-command*. If `=` is used instead of *motion-command* then indent *n* lines (`vip-command-argument`).

*** Call last remembered keyboard macro.

`#` A new vi operator. See Section 1.4.8 [New Commands], page 6, for more details.

The following keys are reserved for future extensions, and currently assigned to a function that just beeps (`vip-nil`).

`&, @, U, [, ], _, q, ~`

VIP uses a special local keymap to interpret key strokes you enter in vi mode. The following keys are bound to *nil* in the keymap. Therefore, these keys are interpreted by the global keymap of Emacs. We give below a short description of the functions bound to these keys in the global keymap. See GNU Emacs Manual for details.

C-@ Set mark and push previous mark on mark ring (`set-mark-command`).

TAB Indent line for current major mode (`indent-for-tab-command`).

C-j Insert a newline, then indent according to mode (`newline-and-indent`).

C-k Kill the rest of the current line; before a newline, kill the newline. With a numeric argument, kill that many lines from point. Negative arguments kill lines backward (`kill-line`).

C-l Clear the screen and reprint everything (`recenter`).

n C-p Move cursor vertically up *n* lines (`previous-line`).

C-q Read next input character and insert it. Useful for inserting control characters (`quoted-insert`).

C-r Search backward incrementally (`isearch-backward`).

C-s Search forward incrementally (`isearch-forward`).

n C-t Interchange characters around point, moving forward one character. With count *n*, take character before point and drag it forward past *n* other characters. If no argument and at end of line, the previous two characters are exchanged (`transpose-chars`).

n C-v Scroll text upward *n* lines. If *n* is not given, scroll near full screen (`scroll-up`).

C-w Kill between point and mark. The text is save in the kill ring. The command *P* or *p* can retrieve it from kill ring (`kill-region`).

2.11 Insert Mode

You can enter insert mode by one of the following commands. In addition to these, you will enter insert mode if you give a change command with a line command as the motion command. Insert commands are also modifying commands and you can repeat them by the repeat command `.` (`vip-repeat`).

i Enter insert mode at point (`vip-insert`).

I Enter insert mode at the first non white character on the line (`vip-Insert`).

a Move point forward by one character and then enter insert mode (`vip-append`).

A Enter insert mode at end of line (`vip-Append`).

o Open a new line below the current line and enter insert mode (`vip-open-line`).

O Open a new line above the current line and enter insert mode (`vip-Open-line`).

C-o Insert a newline and leave point before it, and then enter insert mode (`vip-open-line-at-point`).

Insert mode is almost like emacs mode. Only the following 4 keys behave differently from emacs mode.

⟨ESC⟩ This key will take you back to vi mode (`vip-change-mode-to-vi`).

C-h Delete previous character (`delete-backward-char`).

C-w Delete previous word (`vip-delete-backward-word`).

C-z This key simulates ⟨ESC⟩ key in emacs mode. For instance, typing *C-z x* in insert mode iw the same as typing *ESC x* in emacs mode (`vip-ESC`).

You can also bind *C-h* to `help-command` if you like. (See Section 4.2 [Customizing Key Bindings], page 26, for details.) Binding *C-h* to `help-command` has the effect of making the meaning of *C-h* uniform among emacs, vi and insert modes.

When you enter insert mode, VIP records point as the start point of insertion, and when you leave insert mode the region between point and start point is saved for later use by repeat command etc. Therefore, repeat command will not really repeat insertion if you move point by emacs commands while in insert mode.

3 Ex Commands

In vi mode, you can execute an Ex command *ex-command* by typing:

: ex-command ⟨RET⟩

Every Ex command follows the following pattern:

address command ! parameters count flags

where all parts are optional. For the syntax of *address*, the reader is referred to the reference manual of Ex.

In the current version of VIP, searching by Ex commands is always *magic*. That is, search patterns are always treated as *regular expressions*. For example, a typical forward search would be invoked by *:/pat/*. If you wish to include '/' as part of *pat* you must preceded it by '\'. VIP strips off these \'s before / and the resulting *pat* becomes the actual search pattern. Emacs provides a different and richer class or regular expressions than Vi/Ex, and VIP uses Emacs' regular expressions. See GNU Emacs Manual for details of regular expressions.

Several Ex commands can be entered in a line by separating them by a pipe character '|'.

3.1 Ex Command Reference

In this section we briefly explain all the Ex commands supported by VIP. Most Ex commands expect *address* as their argument, and they use default addresses if they are not explicitly given. In the following, such default addresses will be shown in parentheses.

Most command names can and preferably be given in abbreviated forms. In the following, optional parts of command names will be enclosed in brackets. For example, '`co[py]`' will mean that copy command can be give as '`co`' or '`cop`' or '`copy`'.

If *command* is empty, point will move to the beginning of the line specified by the *address*. If *address* is also empty, point will move to the beginning of the current line.

Some commands accept *flags* which are one of `p`, `l` and `#`. If *flags* are given, the text affected by the commands will be displayed on a temporary window, and you will be asked to hit return to continue. In this way, you can see the text affected by the commands before the commands will be executed. If you hit `C-g` instead of ⟨RET⟩ then the commands will be aborted. Note that the meaning of *flags* is different in VIP from that in Vi/Ex.

(.,.) co[py] addr flags
(.,.) t addr flags
: Place a copy of specified lines after *addr*. If *addr* is `0`, it will be placed before the first line.

(.,.) d[elete] register count flags
: Delete specified lines. Text will be saved in a named *register* if a lower-case letter is given, and appended to a register if a capital letter is given.

`e[dit] !` *+addr file*
`e[x] !` *+addr file*
`vi[sual] !` *+addr file*
: Edit a new file *file* in the current window. The command will abort if current buffer is modified, which you can override by giving `!`. If *+addr* is given, *addr* becomes the current line.

`file` Give information about the current file.

`(1,$) g[lobal] !` */pat/ cmds*
`(1,$) v` */pat/ cmds*
: Among specified lines first mark each line which matches the regular expression *pat*, and then execute *cmds* on each marked line. If `!` is given, *cmds* will be executed on each line not matching *pat*. `v` is same as `g!`.

`(.,.+1) j[oin] !` *count flags*
: Join specified lines into a line. Without `!`, a space character will be inserted at each junction.

`(.) k` *ch*
`(.) mar[k]` *ch*
: Mark specified line by a lower-case character *ch*. Then the addressing form `'`*ch* will refer to this line. No white space is required between `k` and *ch*. A white space is necessary between `mark` and *ch*, however.

`map` *ch rhs* Define a macro for vi mode. After this command, the character *ch* will be expanded to *rhs* in vi mode.

`(.,.) m[ove]` *addr*
: Move specified lines after *addr*.

`(.) pu[t]` *register*
: Put back previously deleted or yanked text. If *register* is given, the text saved in the register will be put back; otherwise, last deleted or yanked text will be put back.

`q[uit] !` Quit from Emacs. If modified buffers with associated files exist, you will be asked whether you wish to save each of them. At this point, you may choose not to quit, by hitting `C-g`. If `!` is given, exit from Emacs without saving modified buffers.

`(.) r[ead]` *file*
: Read in the content of the file *file* after the specified line.

`(.) r[ead] !` *command*
: Read in the output of the shell command *command* after the specified line.

`se[t]` Set a variable's value. See Section 4.1 [Customizing Constants], page 26, for the list of variables you can set.

`sh[ell]` Run a subshell in a window.

`(.,.) s[ubstitute]` */pat/repl/ options count flags*
`(.,.) &` *options count flags*
: On each specified line, the first occurrence of string matching regular expression *pat* is replaced by replacement pattern *repl*. Option characters are `g` and

c. If global option character *g* appears as part of *options*, all occurrences are substituted. If confirm option character *c* appears, you will be asked to give confirmation before each substitution. If */pat/repl/* is missing, the last substitution is repeated.

st[op] Suspend Emacs.

ta[g] tag Find first definition of *tag*. If no *tag* is given, previously given *tag* is used and next alternate definition is find. By default, the file '`TAGS`' in the current directory becomes the *selected tags table*. You can select another tags table by *set* command. See Section 4.1 [Customizing Constants], page 26, for details.

und[o] Undo the last change.

unm[ap] ch
: The macro expansion associated with *ch* is removed.

ve[rsion]
: Tell the version number of VIP.

(1,$) w[rite] ! file
: Write out specified lines into file *file*. If no *file* is given, text will be written to the file associated to the current buffer. Unless *!* is given, if *file* is different from the file associated to the current buffer and if the file *file* exists, the command will not be executed. Unlike Ex, *file* becomes the file associated to the current buffer.

(1,$) w[rite]>> file
: Write out specified lines at the end of file *file*. *file* becomes the file associated to the current buffer.

(1,$) wq ! file
: Same as *write* and then *quit*. If *!* is given, same as *write !* then *quit*.

(.,.) y[ank] register count
: Save specified lines into register *register*. If no register is specified, text will be saved in an anonymous register.

addr ! command
: Execute shell command *command*. The output will be shown in a new window. If *addr* is given, specified lines will be used as standard input to *command*.

($) = Print the line number of the addressed line.

(.,.) > count flags
: Shift specified lines to the right. The variable `vip-shift-width` (default value is 8) determines the amount of shift.

(.,.) < count flags
: Shift specified lines to the left. The variable `vip-shift-width` (default value is 8) determines the amount of shift.

(.,.) ~ options count flags
: Repeat the previous *substitute* command using previous search pattern as *pat* for matching.

The following Ex commands are available in Vi, but not implemented in VIP.

abbreviate, *list*, *next*, *print*, *preserve*, *recover*, *rewind*, *source*, *unabbreviate*, *xit*, *z*

4 Customization

If you have a file called '.vip' in your home directory, then it will also be loaded when VIP is loaded. This file is thus useful for customizing VIP.

4.1 Customizing Constants

An easy way to customize VIP is to change the values of constants used in VIP. Here is the list of the constants used in VIP and their default values.

`vip-shift-width` 8
: The number of columns shifted by `>` and `<` command.

`vip-re-replace` nil
: If `t` then do regexp replace, if `nil` then do string replace.

`vip-search-wrap-around` t
: If `t`, search wraps around the buffer.

`vip-re-search` nil
: If `t` then search is reg-exp search, if `nil` then vanilla search.

`vip-case-fold-search` nil
: If `t` search ignores cases.

`vip-re-query-replace` nil
: If `t` then do reg-exp replace in query replace.

`vip-open-with-indent` nil
: If `t` then indent to the previous current line when open a new line by *o* or *O* command.

`vip-tags-file-name` "TAGS"
: The name of the file used as the tags table.

`vip-help-in-insert-mode` nil
: If `t` then ⟨C-h⟩ is bound to `help-command` in insert mode, if `nil` then it sis bound to `delete-backward-char`.

You can reset these constants in VIP by the Ex command *set*. Or you can include a line like this in your '.vip' file:

```
(setq vip-case-fold-search t)
```

4.2 Customizing Key Bindings

VIP uses `vip-command-mode-map` as the *local keymap* for vi mode. For example, in vi mode, ⟨SPC⟩ is bound to the function `vip-scroll`. But, if you wish to make ⟨SPC⟩ and some other keys behave like Vi, you can include the following lines in your '.vip' file.

```
(define-key vip-command-mode-map "\C-g" 'vip-info-on-file)
(define-key vip-command-mode-map "\C-h" 'vip-backward-char)
(define-key vip-command-mode-map "\C-m" 'vip-next-line-at-bol)
(define-key vip-command-mode-map " " 'vip-forward-char)
```

```
(define-key vip-command-mode-map "g" 'vip-keyboard-quit)
(define-key vip-command-mode-map "s" 'vip-substitute)
(define-key vip-command-mode-map "C" 'vip-change-to-eol)
(define-key vip-command-mode-map "R" 'vip-change-to-eol)
(define-key vip-command-mode-map "S" 'vip-substitute-line)
(define-key vip-command-mode-map "X" 'vip-delete-backward-char)
```

Key Index

0

1

Concept Index

Table of Contents