

```
1: if (false) then
2:   print "False\n";
3: else
4:   print "True\n";
5:
6: int foo = 1234;
7:
8: comment
9:   this is a multi-line comment
10:  TODO: need to do something here
11: end-comment
12:
13: print "Regular code\n";
14: print "String with quote\" in it\n";
15:
16: #define BIG 35
17: #define BIGER 35 + \
18:   44 + identifier
19:
20:
```

```
1: :syntax clear
2: :syntax case ignore
3: :syntax case match
4:
5: :syntax keyword Keyword if then else true false int
6: :syntax keyword Underlined print
7:
8: :syntax match Identifier /[_a-zA-Z][_a-zA-Z0-9]*/
9: :syntax match Number /[0-9]\+/
10:
11: :syntax region Comment start=/comment/ end=/end-comment/
12: :syntax region String start="/" end="/" skip=/"\"/
13:
14: :syntax region Todo start=/TODO:/ end=/$/ contained
15: :syntax region Comment start=/comment/ end=/end-comment/ contains=Todo
16:
17: :syntax region PreProc start=/^#/ end=/$/ oneline
18:
19: " Clear out old syntax so as not to confuse things
20: :syntax clear PreProc
21:
22: :syntax region PreProc start=/^#/ end=/$/ oneline contains=LineContinue
23: :syntax match LineContinue /\\"n.*/* contained
24: :highlight link LineContinue PreProc
```

```

1:
2: hi level1c   ctermfg=brown      guibg=brown      guifg=white
3: hi level2c   ctermfg=Darkblue     guibg=Darkblue   guifg=white
4: hi level3c   ctermfg=darkgray    guibg=darkgray   guifg=white
5: hi level4c   ctermfg=darkgreen   guibg=darkgreen  guifg=white
6: hi level5c   ctermfg=darkcyan    guibg=darkcyan   guifg=white
7: hi level6c   ctermfg=darkred     guibg=darkred    guifg=white
8: hi level7c   ctermfg=darkmagenta  guibg=darkmagenta guifg=white
9: hi level8c   ctermfg=brown      guibg=brown      guifg=white
10: hi level9c  ctermfg=gray       guibg=gray       guifg=white
11: hi level10c ctermfg=black      guibg=black      guifg=white
12: hi level11c ctermfg=darkmagenta guibg=darkmagenta guifg=white
13: hi level12c ctermfg=Darkblue   guibg=Darkblue   guifg=white
14: hi level13c ctermfg=darkgreen  guibg=darkgreen  guifg=white
15: hi level14c ctermfg=darkcyan   guibg=darkcyan   guifg=white
16: hi level15c ctermfg=darkred    guibg=darkred    guifg=white
17: hi level16c ctermfg=red        guibg=red        guifg=white
18:
19:
20:
21: " These are the regions for each pair.
22: " This could be improved, perhaps, by making them match [ and { also,
23: " but I'm not going to take the time to figure out how to make the
24: " end pattern match only the proper type.
25: syn region level1 matchgroup=level1c start=/(/ end=)/ contains=TOP,level1,level
2,level3,level4,level5,level6,level7,level8,level9,level10,level11,level12,level13,level
14,level15,level16,NoInParens
26: syn region level2 matchgroup=level2c start=/(/ end=)/ contains=TOP,level2,level
3,level4,level5,level6,level7,level8,level9,level10,level11,level12,level13,level14,level
15,level16,NoInParens
27: syn region level3 matchgroup=level3c start=/(/ end=)/ contains=TOP,level3,level
4,level5,level6,level7,level8,level9,level10,level11,level12,level13,level14,level15,level
16,NoInParens
28: syn region level4 matchgroup=level4c start=/(/ end=)/ contains=TOP,level4,level
5,level6,level7,level8,level9,level10,level11,level12,level13,level14,level15,level16,
NoInParens
29: syn region level5 matchgroup=level5c start=/(/ end=)/ contains=TOP,level5,level
6,level7,level8,level9,level10,level11,level12,level13,level14,level15,level16,NoInParens
30: syn region level6 matchgroup=level6c start=/(/ end=)/ contains=TOP,level6,level
7,level8,level9,level10,level11,level12,level13,level14,level15,level16,NoInParens
31: syn region level7 matchgroup=level7c start=/(/ end=)/ contains=TOP,level7,level
8,level9,level10,level11,level12,level13,level14,level15,level16,NoInParens
32: syn region level8 matchgroup=level8c start=/(/ end=)/ contains=TOP,level8,level
9,level10,level11,level12,level13,level14,level15,level16,NoInParens
33: syn region level9 matchgroup=level9c start=/(/ end=)/ contains=TOP,level9,level
10,level11,level12,level13,level14,level15,level16,NoInParens
34: syn region level10 matchgroup=level10c start=/(/ end=)/ contains=TOP,level10,level
11,level12,level13,level14,level15,level16,NoInParens
35: syn region level11 matchgroup=level11c start=/(/ end=)/ contains=TOP,level11,level
12,level13,level14,level15,level16,NoInParens
36: syn region level12 matchgroup=level12c start=/(/ end=)/ contains=TOP,level12,level
13,level14,level15,level16,NoInParens
37: syn region level13 matchgroup=level13c start=/(/ end=)/ contains=TOP,level13,level
14,level15,level16,NoInParens
38: syn region level14 matchgroup=level14c start=/(/ end=)/ contains=TOP,level14,level
15,level16,NoInParens
39: syn region level15 matchgroup=level15c start=/(/ end=)/ contains=TOP,level15,level
16,NoInParens
40: syn region level16 matchgroup=level16c start=/(/ end=)/ contains=TOP,level16,NoInParens

```

```
1: "  
2: " Load with gvim -u i-map.vim <file>  
3: "  
4: " Save options  
5: :let s:cpo_save = &cpo  
6: :set cpo&vim  
7:  
8: " Note the Alt version only works in GUI mode  
9: :map <M-i> ^i#include <lt><ESC>A<ESC>j  
10:  
11: :map <F11> ^i#include <lt><ESC>A<ESC>j  
12:  
13: :imap <F11> <ESC>^i#include <lt><ESC>A<  
14:  
15: "  
16: " Menu items only come in if you  
17: " start with gvim -U i-map.vim  
18: "  
19: :menu 40.290 &Tools.&Include<tab>F11 <F11>  
20:  
21: " Restore the previous value of 'coptions'.  
22: :let &cpo = s:cpo_save  
23: :unlet s:cpo_save  
24: :source ~/.gvimrc
```

```
1: " NOTE: gvim -U i-call.vim
2: " Save options
3: :let s:cpo_save = &cpo
4: :set cpo&vim
5: "
6: " Define a function to put in the #include stuff
7: "
8: :function! Include()
9: :   " Get the current line
10: :   let l:line = getline(".")
11:
12: :   " Put the #include in the right place
13: :   let l:line = "#include <".l:line.">"
14:
15: :   " Replace the line
16: :   call setline(".", l:line)
17: :endfunction
18:
19: " Now generate a way of calling the function
20: :map <F11> :call Include()<CR>
21:
22: "NOTE: Define something in the menu
23: :menu 40.290 &Tools.&Include<tab>:call\ Include() :call Include()<CR>
24:
25: "NOTE: Make it work for all modes
26: :amenu 40.290 &Tools.&Include(a)<tab><F11> :call Include()<CR>
27:
28: " NOTE: Define something in the top level menu
29: :amenu 30 &C-Tools.Include<tab>F11 :call Include()<CR>
30:
31: "NOTE: Add it to the popup menu (mouse model)
32: :amenu 1.5 PopUp.&Include<tab>:call\ Include() :call Include()<CR>
33:
34: "NOTE: Put it in the toolbar
35: :amenu icon=/home/sdo/vim/include/include.xpm 1.1 ToolBar.Include :call Include(
) <CR>
36: :tmenu ToolBar.Include Put in the #include line
37:
38: "Enable or disable the menu depending on file type
39: :function CMenuCheck()
40: :   if ((&ft == "c") || (&ft == "cpp"))
41: :       :menu enable &C-Tools
42: :   else
43: :       :menu disable &C-Tools
44: :   endif
45: :endfunction
46:
47: "Automatically call the menu when we enter a buffer or the file type changes
48: :autocmd Bufenter * :call CMenuCheck()
49: :autocmd FileType * :call CMenuCheck()
50:
51: " Restore the previous value of 'coptions'.
52: :let &cpo = s:cpo_save
53: :unlet s:cpo_save
54:
55: :source ~/.gvimrc
```

```
1: " NOTE: gvim -U i-cmd.vim
2: "
3: " Save options
4: :let s:cpo_save = &cpo
5: :set cpo&vim
6:
7: "
8: " Define a function to put in the #include stuff
9: "
10: :function! Include()
11: :   " Get the current line
12: :   let l:line = getline(".")
13: :
14: :   " Put the #include in the right place
15: :   let l:line = "#include <".l:line.">"
16: :
17: :   " Replace the line
18: :   call setline(".", l:line)
19: :endfunction
20:
21:
22: " Simple test function
23: :function! Test(x1, x2)
24: :   echo a:x1." XX ".a:x2
25: :   call getchar()
26: :endfunction
27:
28: :command! -nargs=0 -range TestIt :call Test(<line1>, <line2>)
29:
30: "
31: " Call Include over a range of lines
32: "
33: :function! IncludeRange(first_line, last_line)
34: :   let l:cur_line = a:first_line
35: :
36: :   while (l:cur_line <= a:last_line)
37: :     call setpos('.', [0, l:cur_line, 0, 0])
38: :     call Include()
39: " Debug stuff
40: :echo l:cur_line
41: :redraw
42: :sleep 5
43: :   let l:cur_line = l:cur_line + 1
44: :   endwhile
45: :endfunction
46:
47: :command! -nargs=0 -range Include :call IncludeRange(<line1>, <line2>)
48:
49: :command! -nargs=0 -range Include2 :<line1>, <line2>call Include()
50:
51: " Now generate a way of calling the function
52: :map <F11> :call Include()<CR>
53:
54: "NOTE: Define something in the menu
55: :menu 40.290 &Tools.&Include<tab>:call\ Include() :call Include()<CR>
56:
57: "NOTE: Make it work for all modes
58: :amenu 40.295 &Tools.&Include(a)<tab>:call\ Include() :call Include()<CR>
59:
60: " NOTE: Define something in the top level menu
61: :amenu 30 &C-Tools.Include :call Include()<CR>
62:
63: "NOTE: Add it to the popup menu (mouse model)
64: :amenu 1.5 PopUp.&Include<tab>:call\ Include() :call Include()<CR>
65:
```

```
66:"NOTE: Put it in the toolbar
67:amenu icon=/home/sdo/vim/include/include.xpm 1.1 ToolBar.Include :call Include(
)<CR>
68:tmenu ToolBar.Include Put in the #include line
69:
70:"Enable or disable the menu depending on file type
71:function! CMenuCheck()
72:    if (&ft == "c") || (&ft == "cpp")
73:        menu enable &C-Tools
74:    else
75:        menu disable &C-Tools
76:    endif
77:endfunction
78:
79:"Automatically call the menu when we enter a buffer or the file type changes
80:autocmd Bufenter * :call CMenuCheck()
81:autocmd FileType * :call CMenuCheck()
82:
83:filetype on
84:" Restore the previous value of 'coptions'.
85:let &cpo = s:cpo_save
86:unlet s:cpo_save
87:source ~/.gvimrc
```

```
1: /* XPM */
2: static char * include_xpm[] = {
3: "18 18 3 1",
4: "      c None",
5: ".      c #FFFFFF",
6: "+      c #FF0000",
7: ". . . . .+ . . . . .",
8: ". . . . .+.+. . . . .",
9: ". . . . .+ . . . . .",
10: ". . . . .+ . . . . .",
11: ". . . . .+. . . . . .",
12: ". . . . .+ . . . . .",
13: ". . . . .+. . . . . .",
14: ". . . . .+. . . . . .",
15: ". . . . .+. . . . . .",
16: ". . . . .+. . . . . .",
17: ". . . . .+. . . . . .",
18: ". . . . .+. . . . . .",
19: ". . . . .+. . . . . .",
20: ". . . . .+. . . . . .",
21: ". . . . .+. . . . . .",
22: ". . . . .+. . . . . .",
23: ". . . . .+. . . . . .",
24: ". . . . .+. . . . . ."};
```

```
1: " NOTE: Start with gvim -U i-fancy.vim
2: " Save options
3: :let s:cpo_save = &cpo
4: :set cpo&vim
5: "
6: " Configuration section follow
7: "
8: " System include directories
9: :let g:SystemIncludes = [
10:\    "/usr/include",
11:\    "/usr/include/gnu",
12:\    "/usr/include/net",
13:\    "/usr/include/linux",
14:\    "/usr/include/sys",
15:\    "/usr/include/c++/3.4.3"
16:\]
17:
18:" Local includes follow
19: :let g:LocalIncludes = [
20:\    ".",
21:\    "include"
22:\]
23:"
24:" Define a function to put in the #include stuff
25:"
26: :function! Include()
27: :    " Get the current line
28: :    let l:line = getline(".")
29: :
30: :    " Loop through the local directories looking for the file
31: :    for l:cur_dir in g:LocalIncludes
32: :        if (filereadable(l:cur_dir."/".l:line))
33: :            " Put the #include in the right place
34: :            let l:line = "#include \"".l:line."\"
35: :            call setline(".", l:line)
36: :            return
37: :        endif
38: :    endfor
39: :
40: :    " Loop through the system directories looking for the file
41: :    for l:cur_dir in g:SystemIncludes
42: :        if (filereadable(l:cur_dir."/".l:line))
43: :            " Put the #include in the right place
44: :            let l:line = "#include <".l:line.">"
45: :            call setline(".", l:line)
46: :            return
47: :        endif
48: :    endfor
49: :
50: :    " Put in a default #include
51: :    let l:line = "#include \"".l:line."\" // WARNING: Can not locate include"
52: :
53: :    " Replace the line
54: :    call setline(".", l:line)
55: :endfunction
56:
57:" Now generate a way of calling the function
58: :map <F11> :call Include()<CR>
59:
60: :filetype on
61:" Restore the previous value of 'cpoptions'.
62: :let &cpo = s:cpo_save
63: :unlet s:cpo_save
64: :source ~/.gvimrc
```

```
1: "  
2: " NOTE: Start with gvim -U i-path.vim  
3: "  
4: " Save options  
5: :let s:cpo_save = &cpo  
6: :set cpo&vim  
7: "  
8: " Define a function to put in the #include stuff  
9: "  
10: :function! Include()  
11: :     " Get the current line  
12: :     let l:line = getline(".")  
13: :  
14: :     let l:dir_list = split(&path, ",")  
15: :     " Loop through the local directories looking for the file  
16: :     for l:cur_dir in l:dir_list  
17: :         if (filereadable(l:cur_dir."/.l:line))  
18: :  
19: :             " System directory?  
20: :             if (match(l:cur_dir, "/usr/include") == 0)  
21: :  
22: :                 " Put the #include in the right place  
23: :                 let l:line = "#include <".l:line.">"  
24: :             else  
25: :                 " Put the #include in the right place  
26: :                 let l:line = "#include \"".l:line."\""  
27: :             endif  
28: :  
29: :             call setline(".", l:line)  
30: :             return  
31: :         endif  
32: :     endfor  
33: :  
34: :     "At this point we did not find anything  
35: :     "We could put in a default  
36: :endfunction  
37: :  
38: " Now generate a way of calling the function  
39: :map <F11> :call Include()<CR>  
40: :  
41: " Restore the previous value of 'coptions'.  
42: :let &cpo = s:cpo_save  
43: :unlet s:cpo_save  
44: :source ~/.gvimrc
```

```
1: "  
2: " NOTE: Start with gvim -u i-path.vim  
3: "  
4: " Save options  
5: :let s:cpo_save = &cpo  
6: :set cpo&vim  
7: "  
8: " Define a function to put in the #include stuff  
9: "  
10: :function! Include()  
11: :     " Get the current line  
12: :     let l:line = getline(".")  
13: :  
14: :     let l:choice = confirm(  
15: :         "What type of #include is ".l:line."?",  
16: :         "&System\n&Local\nNo \&Change")  
17: :  
18: :     if (l:choice == 1)  
19: :         let l:line = "#include <".l:line.">"  
20: :         call setline(".", l:line)  
21: :         return  
22: :     elseif (l:choice == 2)  
23: :         let l:line = "#include \\".l:line.\\""  
24: :         call setline(".", l:line)  
25: :         return  
26: :     elseif (l:choice == 3)  
27: :         return  
28: :     elseif (l:choice == 0)  
29: :         throw "WARNING: You closed the dialog!"  
30: :     else  
31: :         throw "ERROR: There is no choice ".l:choice." Huh?"  
32: :     endif  
33: :endfunction  
34: :  
35: " Now generate a way of calling the function  
36: :map <F11> :call Include()<CR>  
37: :  
38: " Restore the previous value of 'coptions'.  
39: :let &cpo = s:cpo_save  
40: :unlet s:cpo_save  
41: :source ~/.gvimrc
```

```
1: class bean {  
2:     private int foo;  
3: }
```

```

1: "
2: " NOTE: This does not properly handle types with
3: "       dots in them like
4: "
5: "       private class.type foo;
6: "
7: " We could fix it but for this demonstration the regular
8: " expression are already nasty enough
9: "
10::function! Getter()
11::    " Get the line defining the variable
12::    let l:var_line = getline('.')
13::    let l:prot = substitute(l:var_line, '\v^\W*(\w+)\W+.*$', '\1', '')
14::    let l:type = substitute(l:var_line, '\v^\W*\w+\W+(\w+)\W+.*$', '\1', '')
15::    let l:var = substitute(l:var_line, '\v^\W*\w+\W+\w+\W+(\w+).*', '\1', '')
16::    if ((l:prot != 'public') && (l:prot != 'private') && (l:prot != 'protected'
))
17::        throw "ERROR: Unable to parse variable line"
18::    endif
19::    " Move cursor back to the class keyword
20::    if (search('class', 'b') == 0)
21::        throw "ERROR: Could not find class line"
22::    endif
23::    " Move cursor forward to the { after the class
24::    if (search('{', '') == 0)
25::        throw "ERROR: Could not find class line"
26::    endif
27::    Go to matching {
28::    %
29::    let l:fun_name = substitute(l:var, '.*', '\u&', '')
30::    let l:getter = [
31:\        "    /*",
32:\        "    * Get the current value of ".l:var,
33:\        "    * ",
34:\        "    * @returns ".l:var,
35:\        "    */",
36:\        "    public ".l:type." get".l:fun_name."() {",
37:\        "        return(".l:var.");",
38:\        "    }"
39:\    ]
40::    let l:where = line('.')
41::    let l:where = l:where - 1
42::    call append(l:where, l:getter)
43::endfunction
44:
45::map <F11> :call Getter(<cr>
46:
47::source ~/.gvimrc

```

```
1: class mean {  
2:     /* class in a comment */  
3:     private final string nasty = "class in a string"  
4:     private int foo;  
5: }
```

```

1: "
2: " NOTE: This does not properly handle types with
3: "     dots in them like
4: "
5: "     private class.type foo;
6: "
7: " We could fix it but for this demonstration the regular
8: " expression are already nasty enough
9: "
10::function! Getter()
11::    " Get the line defining the variable
12::    let l:var_line = getline('.')
13::    let l:prot = substitute(l:var_line, '\v^\W*(\w+)\W+.*$', '\1', '')
14::    let l:type = substitute(l:var_line, '\v^\W*\w+\W+(\w+)\W+.*$', '\1', '')
15::    let l:var = substitute(l:var_line, '\v^\W*\w+\W+\w+\W+(\w+).*$', '\1', '')
16::    if ((l:prot != 'public') && (l:prot != 'private') && (l:prot != 'protected'
))
17::        throw 'ERROR: Unable to parse variable line'
18::    endif
19::
20::    " Loop until we find a java class declaration
21::    while (1)
22::        " Move cursor back to the class keyword
23::        if (search('class', 'b') == 0)
24::            throw 'ERROR: Could not find class line'
25::        endif
26::        if (synIDattr(synID(line('.'), col('.'), 1), 'name') ==
27:\                'javaClassDecl')
28::            break
29::        endif
30::    endwhile
31::
32::    while (1)
33::        " Move cursor forward to the { after the class
34::        if (search('{', '') == 0)
35::            throw 'ERROR: Could not find class line'
36::        endif
37::        if (synIDattr(synID(line('.'), col('.'), 1), 'name') == '')
38::            break
39::        endif
40::    endwhile
41::    Go to matching {
42::    %
43::    let l:fun_name = substitute(l:var, '.*', '\u&', '')
44::    let l:getter = [
45:\        '        * Get the current value of '.l:var,
46:\        '        *',
47:\        '        * @returns '.l:var,
48:\        '        */',
49:\        '        public '.l:type.' get'.l:fun_name.'() {',
50:\        '            return('.l:var. ');',
51:\        '        }',
52:\        ]
53:
54::    let l:where = line('.')
55::    let l:where = l:where - 1
56::    call append(l:where, l:getter)
57::endfunction
58:
59::map <F11> :call Getter()<cr>
60:
61::source ~/.gvimrc

```

```

1: use strict;
2: use warnings;
3: use constant FUDGE => 2;
4:
5: sub make_tab(@)
6: {
7:     my @lines = @_;      # The lines to split up
8:     my @data;           # The split apart lines
9:     my @size;           # Sizes of the columns
10:
11:     foreach my $line (@lines) {
12:         # Split the line into words
13:         my @words = split /\s+/, $line;
14:
15:         push(@data, [@words]);
16:         for (my $i = 0; $i <= $#words; ++$i) {
17:             if (defined($size[$i])) {
18:                 if (length($words[$i]) <= ($size[$i]- FUDGE)) {
19:                     next;
20:                 }
21:             }
22:
23:             $size[$i] = length($words[$i]) + FUDGE;
24:         }
25:     }
26:     my @result; # resulting line
27:     foreach my $cur_line (@data) {
28:         my $new_line = ""; # Line we are building up
29:
30:         for (my $i = 0; $i <= $#$cur_line; ++$i) {
31:             my $width = $size[$i]; # Size of this col
32:             my $word = $cur_line->[$i];
33:             while (length($word) < $width) {
34:                 $word .= ".";
35:             }
36:             $new_line .= $word;
37:         }
38:         push (@result, $new_line);
39:     }
40:     return (@result);
41: }
42: 1;

```

```
1: #
2: # Program to fix a table
3: #
4: use strict;
5: use warnings;
6: use tab;
7:
8: my @lines = <>;
9: chomp(@lines);
10: @lines = make_tab(@lines);
11:
12: foreach my $line (@lines) {
13:     print "$line\n";
14: }
15:
```

```
1: "  
2: " Perl script to turn text into a table  
3: "  
4: "  
5: " Make sure we have the perl feature  
6: :if ! has('perl')  
7: :   throw "ERROR: This version of Vim has no Perl feature"  
8: :endif  
9:  
10: "  
11: " Define the function to do the work  
12: "  
13: :perl <<EOF  
14: # Real work done here  
15: require 'tab.pm';  
16:  
17: sub tab_lines($$) {  
18:     my $start = shift;  
19:     my $end = shift;  
20:     my $cur_buf = $main::curbuf;  
21:  
22:     my @lines = $cur_buf->Get($start..$end);  
23:     @lines = make_tab(@lines);  
24:     $cur_buf->Set($start, @lines);  
25: }  
26: EOF  
27:  
28: :command! -nargs=0 -range Table :perl tab_lines(<line1>, <line2>)
```