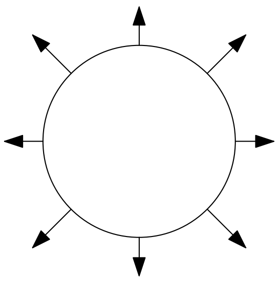
Examples of `pic` Macros

W. Richard Stevens

This document contains examples of the `pic` macros used in my books. Using `pic` requires developing a set of macros to do common tasks. These macros have been developed and tweaked over the past 7 years. Many of the macros, especially the splines, were developed by Gary Wright.

The following shows the compass points on a circle.

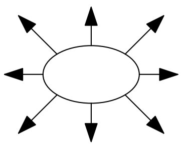
```
A:   circle rad 0.5
     arrow up 0.2 from A.n
     arrow up 0.2 right 0.2 from A.ne
     arrow right 0.2 from A.e
     arrow down 0.2 right 0.2 from A.se
     arrow down 0.2 from A.s
     arrow down 0.2 left 0.2 from A.sw
     arrow left 0.2 from A.w
     arrow up 0.2 left 0.2 from A.nw
     " check the compass points on a circle" at A.e + (0.3, 0) ljust
```



check the compass points on a circle

And the compass points on an ellipse.

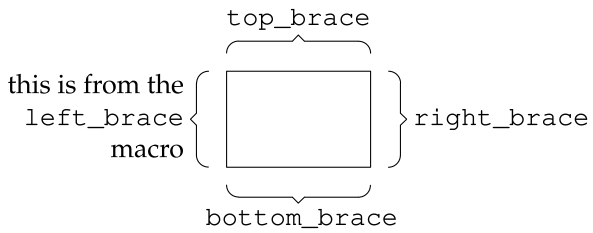
```
A:   ellipse wid 0.5 ht 0.3
     arrow up 0.2 from A.n
     arrow up 0.2 right 0.2 from A.ne
     arrow right 0.2 from A.e
     arrow down 0.2 right 0.2 from A.se
     arrow down 0.2 from A.s
     arrow down 0.2 left 0.2 from A.sw
     arrow left 0.2 from A.w
     arrow up 0.2 left 0.2 from A.nw
     " check the compass points on a ellipse" at A.e + (0.3, 0) ljust
```



check the compass points on a ellipse

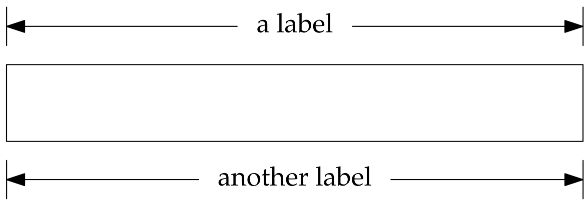
I use four macros to draw braces in all our directions. Each macro defines the location `Bracept` and text can be placed there after drawing the brace. When multiple lines of text are needed at the `Bracept` I draw a box of width 0.1, position the box at the `Bracept` and then either left justify or right justify each line. Note that I use the `left_space` macro to add 2 inches of spacing to the left of the picture. This is because the text items have no width.

```
B:   box at (2.0, 0)
     right_brace( B.ne, B.se )
          " \fCright_brace\fP" at Bracept ljust
     left_brace( B.nw, B.sw )
          box invis wid 0.1 with .c at Bracept \
               "this is from the " rjust \
               "\fCleft_brace\fP " rjust \
               "macro " rjust
     left_space(Bracept, 2.0)
     top_brace( B.nw, B.ne )
          "\fCtop_brace\fP" at Bracept above
     bottom_brace( B.sw, B.se )
          " \fCbottom_brace\fP" at Bracept below
```



The two macros `label_above` and `label_below` draw labels above or below a box. You have to measure the lines to the right and left of the arrows, to leave enough space for the text in the middle.

```
A:   box wid 3.0 ht 0.4
     label_above( A.nw, A.ne, 1.2, "a label" )
     label_below( A.sw, A.se, 1.0, "another label" )
```

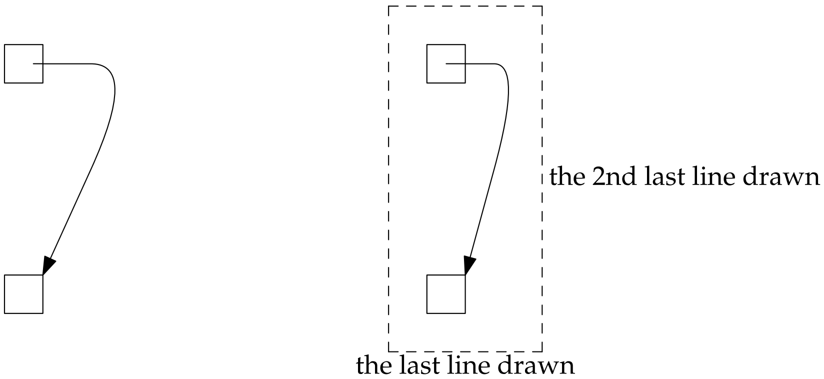


The `line_gap_down` and `line_gap_right` macros draw a line with a squiggle in the middle.

```
A:   line right 0.2
B:   line right 0.2 with .start at A.start + (0, -0.8)
     line_gap_down( A.c, B.c )

C:   line down 0.2 with .start at A.end + (2.0, 0)
D:   line down 0.2 with .start at C.start + (1.0, 0)
     line_gap_right( C.c, D.c )
```

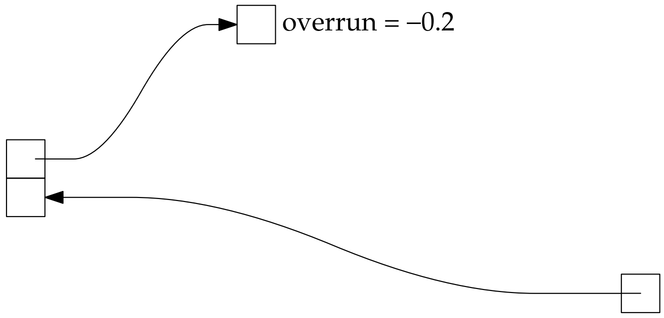
This is the `vv_spline` macro. The `dash_box` macro draws a dashed box around anything, given the lower-left and upper-right locations.

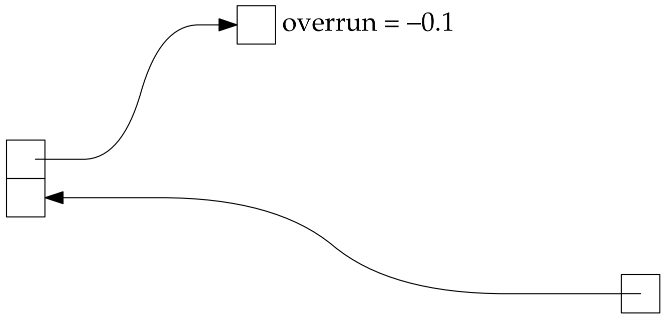
```
A:   box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .n at A.s + (0, -1.0)
     vv_spline( A.e, B.ne, -0.05, 0.5, -> )

C:   box ht 0.2 wid 0.2 with .w at A.e + (2.0, 0)
D:   box ht 0.2 wid 0.2 with .n at C.s + (0, -1.0)
     vv_spline( C.e, D.ne, -0.10, 0.3, -> )
     dash_box( D.sw, C.ne + (0.2, 0) )
          "the last line drawn" at last line.c below
          " the 2nd last line drawn" at 2nd last line.c ljust
```

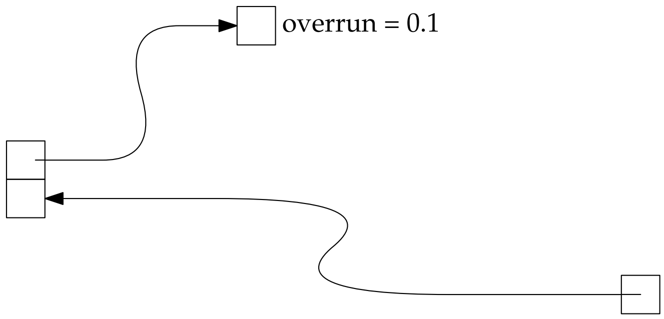


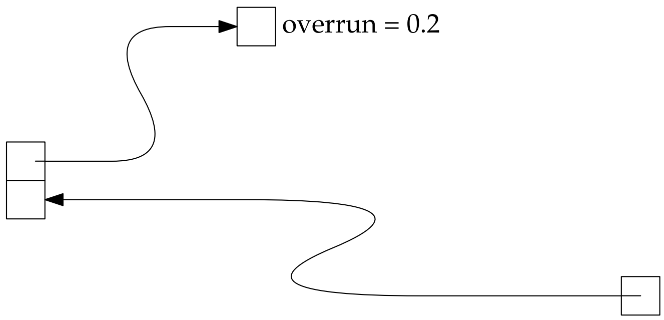
The `hz_spline` macro with some different values for the `overrun` argument.

```
     down
A1:  box ht 0.2 wid 0.2
A2:  box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .sw at A1.ne + (1.0,  0.5)
C:   box ht 0.2 wid 0.2 with .nw at A1.se + (3.0, -0.5)
     hz_spline( A1.e, B.w, 0.05, 0, ->, 0.5, -0.2 )
     hz_spline( A2.e, C.w, 0, 0.10, <-, 0.5, -0.2 )
     " overrun = \(mi0.2" at B.e ljust

     down
A1:  box ht 0.2 wid 0.2 with .n at A1.s + (0, -1.7)
A2:  box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .sw at A1.ne + (1.0,  0.5)
C:   box ht 0.2 wid 0.2 with .nw at A1.se + (3.0, -0.5)
     hz_spline( A1.e, B.w, 0.05, 0, ->, 0.5, -0.1 )
     hz_spline( A2.e, C.w, 0, 0.10, <-, 0.5, -0.1 )
     " overrun = \(mi0.1" at B.e ljust

     down
A1:  box ht 0.2 wid 0.2 with .n at A1.s + (0, -1.7)
A2:  box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .sw at A1.ne + (1.0,  0.5)
C:   box ht 0.2 wid 0.2 with .nw at A1.se + (3.0, -0.5)
     hz_spline( A1.e, B.w, 0.05, 0, ->, 0.5, 0.1 )
     hz_spline( A2.e, C.w, 0, 0.10, <-, 0.5, 0.1 )
     " overrun = 0.1" at B.e ljust

     down
A1:  box ht 0.2 wid 0.2 with .n at A1.s + (0, -1.7)
A2:  box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .sw at A1.ne + (1.0,  0.5)
C:   box ht 0.2 wid 0.2 with .nw at A1.se + (3.0, -0.5)
     hz_spline( A1.e, B.w, 0.05, 0, ->, 0.5, 0.2 )
     hz_spline( A2.e, C.w, 0, 0.10, <-, 0.5, 0.2 )
     " overrun = 0.2" at B.e ljust
```


overrun = –0.2

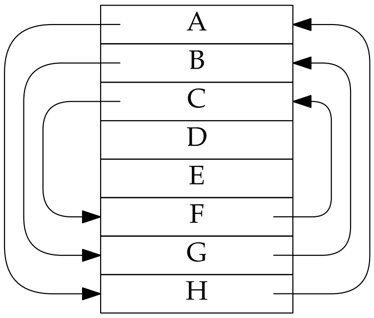

overrun = –0.1


overrun = 0.1


overrun = 0.2

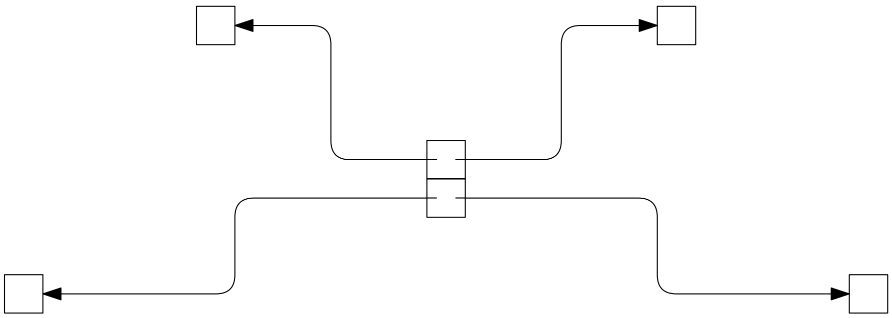
The `spline_dl` draws a spline down on the left side, and the `spline_ur` goes up on the right side. There are also `spline_dr` and `spline_ul` macros.

```
     down
A:   box ht 0.2 wid 1.0 "A"
B:   box ht 0.2 wid 1.0 "B"
C:   box ht 0.2 wid 1.0 "C"
D:   box ht 0.2 wid 1.0 "D"
E:   box ht 0.2 wid 1.0 "E"
F:   box ht 0.2 wid 1.0 "F"
G:   box ht 0.2 wid 1.0 "G"
H:   box ht 0.2 wid 1.0 "H"
     spline_dl( A.w, H.w,  0.1, 0.5, -> )
     spline_dl( B.w, G.w,  0.1, 0.4, -> )
     spline_dl( C.w, F.w,  0.1, 0.3, -> )
     spline_ur( H.e, A.e, -0.1, 0.4, -> )
     spline_ur( G.e, B.e, -0.1, 0.3, -> )
     spline_ur( F.e, C.e, -0.1, 0.2, -> )
```



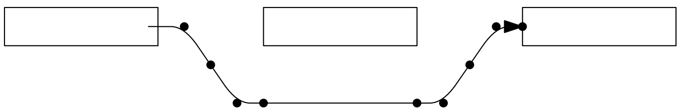
The following four demonstrate macros right-up-right, right-down-right, left-up-left, and left-down-left.

```
     down
A1:  box ht 0.2 wid 0.2
A2:  box ht 0.2 wid 0.2
B:   box ht 0.2 wid 0.2 with .sw at A1.ne + ( 1.0,  0.5)
C:   box ht 0.2 wid 0.2 with .nw at A1.se + ( 2.0, -0.5)
D:   box ht 0.2 wid 0.2 with .se at A1.nw + (-1.0,  0.5)
E:   box ht 0.2 wid 0.2 with .ne at A1.sw + (-2.0, -0.5)
     spline_rur( A1.e, B.w, -0.05, 0, -> )
     spline_rdr( A2.e, C.w, -0.05, 0, -> )
     spline_lul( A1.w, D.e,  0.05, 0, -> )
     spline_ldl( A2.w, E.e,  0.05, 0, -> )
```



Sometimes I need a custom spline when one of the macros just shown does not fit the job. This example is from page 322 of UNPv1. When making a curve I divide the distance into quarters, with one move per quarter.

```
boxwid = 0.8
boxht = 0.2
Box1:    box
Box2:    box with .w at Box1.e + (0.55, 0)
Box3:    box with .w at Box2.e + (0.55, 0)

Start:   Box1.e
Dest:    Box3.w
     line left 0.05 from Start
        spline -> from Start \
                right 0.55/4 \
                then right 0.55/4 down 0.2 \
                then right 0.55/4 down 0.2 \
                then right 0.55/4 \
                then right boxwid \
                then right 0.55/4 \
                then right 0.55/4 up 0.2 \
                then right 0.55/4 up 0.2 \
                then right 0.55/4
# Draw the "points" in this spline definition.
A:   line invis right 0.55/4 from Start
B:   line invis right 0.55/4 down 0.2
C:   line invis right 0.55/4 down 0.2
D:   line invis right 0.55/4
E:   line invis right boxwid
F:   line invis right 0.55/4
G:   line invis right 0.55/4 up 0.2
H:   line invis right 0.55/4 up 0.2
I:   line invis right 0.55/4
     circle rad 0.02 fill 1 at A.end
     circle rad 0.02 fill 1 at B.end
     circle rad 0.02 fill 1 at C.end
     circle rad 0.02 fill 1 at D.end
     circle rad 0.02 fill 1 at E.end
     circle rad 0.02 fill 1 at F.end
     circle rad 0.02 fill 1 at G.end
     circle rad 0.02 fill 1 at H.end
     circle rad 0.02 fill 1 at I.end
```



The `erase` macro is used to white out some portion of a figure when lines cross. I first specify a second argument of 1 (draw the circle all black) to check the position of the circle, then change the argument to 0 to erase the circle.

```
     right
A:   box dashed
B:   box dashed
     line from A.nw to A.se
     erase(A.c, 1, 0.05)     # draw a black circle
     line from A.sw to A.ne
     line from B.nw to B.se
     erase(B.c, 0, 0.05)     # draw a white circle
     line from B.sw to B.ne
```