**Software Change Request**

| **Software** | **Baseline Version** | **Feature Name** | **Difficulty** |
| --- | --- | --- | --- |
| Xfig | 3.2.8a | Free Selection | Medium (Est. 5 files; 640 LOC) |

***Current Behaviors:***

Currently Xfig allows a user to select a single object and then move it. However, this is not the most efficient when the user has to move multiple objects while keeping their relative locations to each other.

***Expected Behavior:***

Allow a user to create a closed-loop shape that will select all objects within the shape. In the given solution, the chosen shape is a rectangle, but in practice this shape could be an ellipse, or a regular n-gon. Holding and dragging the left mouse button will move all selected objects simultaneously while keeping their relative positions unchanged. Releasing the left mouse button will determine the new location. Also add a new shortcut button in the Editing Mode Panel on the left hand side of Xfig. This procedure is demonstrated in Figure 1 below.

Graphical user interface, application

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

*Figure 1: the procedure of using the free selection feature (left: create a freeform selection shape through the shortcut button, middle: move the selected objects within the freeform selection shape, right: the new location for the selected objects)*

***Solution Hints for Instructor:***

This freeform selection can be applied to many other actions, such as scaling and attribute editing. The example solution only implements the selection as a box, but the principle should be the same for more complicated shapes. First, all drawable objects within the selection area have their **tagged** internal variables set. Then, these tagged shapes are added to a compound to be moved or deleted. Most of the code is implemented in two new files (e\_ffst.c and e\_ffst.h), with the rest consisting of minor changes to existing files.

An additional challenge could be to make the selection area completely freeform. The selection area could be drawn using the built-in freehand line draw, with the start and end points being automatically connected with a line. The difficulty in this task could come from developing an efficient algorithm to tag all objects inside the area.

**e\_ffst.c:**

#include "fig.h"

#include "resources.h"

#include "object.h"

#include "mode.h"

#include "paintop.h"

#include "d\_text.h"

#include "u\_create.h"

#include "u\_elastic.h"

#include "u\_list.h"

#include "u\_draw.h"

#include "u\_drag.h"

#include "u\_search.h"

#include "w\_canvas.h"

#include "w\_drawprim.h"

#include "w\_indpanel.h"

#include "w\_mousefun.h"

#include "w\_msgpanel.h"

#include "w\_setup.h"

#include "w\_util.h"

#include "w\_layers.h"

#include "e\_ffst.h"

#include "e\_scale.h"

#include "f\_read.h"

#include "f\_util.h"

#include "u\_bound.h"

#include "u\_fonts.h"

#include "u\_free.h"

#include "u\_markers.h"

#include "u\_redraw.h"

#include "u\_undo.h"

#include "w\_cursor.h"

#include "w\_util.h"

//------------------------Base Code--------------------------------------------------------------------------------

F\_compound \*temp = NULL;

void ffst\_selected(void)

{

set\_mousefun("corner point", "", "", "", "", "");

canvas\_kbd\_proc = null\_proc;

canvas\_locmove\_proc = null\_proc;

canvas\_leftbut\_proc = init\_border\_drawing;

canvas\_middlebut\_proc = null\_proc;

canvas\_rightbut\_proc = null\_proc;

set\_cursor(crosshair\_cursor);

reset\_action\_on();

}

void

init\_border\_drawing(int x, int y)

{

cur\_x = fix\_x = x;

cur\_y = fix\_y = y;

set\_mousefun("final point", "", "cancel", "", "", "");

draw\_mousefun\_canvas();

//set mouse actions

canvas\_locmove\_proc = resizing\_box;

canvas\_leftbut\_proc = create\_selectArea;

canvas\_middlebut\_proc = null\_proc;

canvas\_rightbut\_proc = cancel\_box;

//create box to draw

elastic\_box(fix\_x, fix\_y, cur\_x, cur\_y);

set\_cursor(null\_cursor);

set\_action\_on();

}

static void

cancel\_box(void)

{

elastic\_box(fix\_x, fix\_y, cur\_x, cur\_y);

/\* erase last lengths if appres.showlengths is true \*/

erase\_box\_lengths();

ffst\_selected();

draw\_mousefun\_canvas();

}

static void

create\_selectArea(int x, int y)

{

elastic\_box(fix\_x, fix\_y, cur\_x, cur\_y);

int x\_min, x\_max, y\_min, y\_max;

x\_min = min2(fix\_x, x);

y\_min = min2(fix\_y, y);

x\_max = max2(fix\_x, x);

y\_max = max2(fix\_y, y);

tag\_obj\_in\_region1(x\_min, y\_min, x\_max, y\_max);

//Create the compound

create\_compoundobject(1, 1);

draw\_mousefun\_canvas();

}

static void

break\_comp(void)

{

init\_break();

ffst\_selected();

}

//---------------------------GLUE CODE-------------------------------------------------------------------------------- --

//---------------------------------------------------------------------------------- ------------------------------------

static void

create\_compoundobject(int x, int y)

{

(void)x; (void)y;

F\_compound\* c;

if ((c = create\_compound()) == NULL)

return;

//dont create compound if nothing is selected

if (compose\_compound1(c) == 0) {

free((char\*)c);

ffst\_selected();

draw\_mousefun\_canvas();

put\_msg("Empty compound, ignored");

return;

}

/\*

\* Make the bounding box exactly match the dimensions of the compound. \*/

compound\_bound(c, &c->nwcorner.x, &c->nwcorner.y,

&c->secorner.x, &c->secorner.y);

/\* if zero width or height in the compound, adjust to next positioning grid point or a few pixels if positioning grid is "ANY" \*/

if (c->nwcorner.x == c->secorner.x)

{

if (cur\_pointposn != P\_ANY)

{

c->secorner.x += point\_spacing();

c->secorner.x = ceil\_coords\_x(c->secorner.x, c->secorner.y);

}

}

if (c->nwcorner.y == c->secorner.y)

{

if (cur\_pointposn != P\_ANY)

{

c->secorner.y += point\_spacing();

c->secorner.y = ceil\_coords\_y(c->secorner.x, c->secorner.y);

}

}

//cleanup for creating new compound

c->next = NULL;

clean\_up();

set\_action(F\_GLUE);

toggle\_markers\_in\_compound(c);

list\_add\_compound(&objects.compounds, c);

mask\_toggle\_compoundmarker(c);

set\_latestcompound(c);

temp = c;

set\_modifiedflag();

move\_selected1(); //Allows for clean progression of functions draw\_mousefun\_canvas();

}

int compose\_compound1(F\_compound\* c)

{

c->ellipses = NULL;

c->lines = NULL;

c->texts = NULL;

c->splines = NULL;

c->arcs = NULL;

c->comments = NULL;

c->compounds = NULL;

/\* defer updating of layer buttons until we've composed the entire compound \*/

defer\_update\_layers = True;

get\_ellipse(&c->ellipses);

get\_line(&c->lines);

get\_spline(&c->splines);

get\_text(&c->texts);

get\_arc(&c->arcs);

get\_compound(&c->compounds);

/\* now update the layer buttons \*/

defer\_update\_layers = False;

update\_layers();

if (c->ellipses != NULL)

return (1);

if (c->splines != NULL)

return (1);

if (c->lines != NULL)

return (1);

if (c->texts != NULL)

return (1);

if (c->arcs != NULL)

return (1);

if (c->compounds != NULL)

return (1);

return (0);

}

void

tag\_obj\_in\_region1(int xmin, int ymin, int xmax, int ymax)

{

sel\_ellipse(xmin, ymin, xmax, ymax);

sel\_line(xmin, ymin, xmax, ymax);

sel\_spline(xmin, ymin, xmax, ymax);

sel\_text(xmin, ymin, xmax, ymax);

sel\_arc(xmin, ymin, xmax, ymax);

sel\_compound(xmin, ymin, xmax, ymax);

}

static void

sel\_ellipse(int xmin, int ymin, int xmax, int ymax)

{

F\_ellipse\* e;

for (e = objects.ellipses; e != NULL; e = e->next) {

if (!active\_layer(e->depth))

continue;

if (xmin > e->center.x - e->radiuses.x)

continue;

if (xmax < e->center.x + e->radiuses.x)

continue;

if (ymin > e->center.y - e->radiuses.y)

continue;

if (ymax < e->center.y + e->radiuses.y)

continue;

//tag ellipse as object

e->tagged = 1 - e->tagged;

toggle\_ellipsehighlight(e);

}

}

static void

get\_ellipse(F\_ellipse\*\* list)

{

F\_ellipse\* e, \* ee, \* ellipse;

for (e = objects.ellipses; e != NULL;)

{

if (!e->tagged)

{

ee = e;

e = e->next;

continue;

}

remove\_depth(O\_ELLIPSE, e->depth);

if (\*list == NULL)

\*list = e;

else

ellipse->next = e;

ellipse = e;

if (e == objects.ellipses)

e = objects.ellipses = objects.ellipses->next;

else

e = ee->next = e->next;

ellipse->next = NULL;

}

}

static void

sel\_arc(int xmin, int ymin, int xmax, int ymax)

{

F\_arc\* a;

int urx, ury, llx, lly;

for (a = objects.arcs; a != NULL; a = a->next)

{

if (!active\_layer(a->depth))

continue;

arc\_bound(a, &llx, &lly, &urx, &ury);

if (xmin > llx)

continue;

if (xmax < urx)

continue;

if (ymin > lly)

continue;

if (ymax < ury)

continue;

a->tagged = 1 - a->tagged;

toggle\_archighlight(a);

}

}

static void

get\_arc(F\_arc\*\* list)

{

F\_arc\* a, \* arc, \* aa;

for (a = objects.arcs; a != NULL;)

{

//iterate until we find a selected arc

if (!a->tagged)

{

aa = a;

a = a->next;

continue;

}

remove\_depth(O\_ARC, a->depth);

if (\*list == NULL)

\*list = a;

else

arc->next = a;

arc = a;

if (a == objects.arcs)

a = objects.arcs = objects.arcs->next;

else

a = aa->next = a->next;

arc->next = NULL;

}

}

static void

sel\_line(int xmin, int ymin, int xmax, int ymax) {

F\_line\* l;

F\_point\* p;

int inbound;

for (l = objects.lines; l != NULL; l = l->next)

{

if (!active\_layer(l->depth))

continue;

for (inbound = 1, p = l->points; p != NULL && inbound; p = p->next)

{

inbound = 0;

if (xmin > p->x)

continue;

if (xmax < p->x)

continue;

if (ymin > p->y)

continue;

if (ymax < p->y)

continue;

inbound = 1;

}

if (!inbound)

continue;

l->tagged = 1 - l->tagged;

toggle\_linehighlight(l);

}

}

static void

get\_line(F\_line\*\* list)

{

F\_line\* line, \* l, \* ll;

for (l = objects.lines; l != NULL;)

{

if (!l->tagged)

{

ll = l;

l = l->next;

continue;

}

remove\_depth(O\_POLYLINE, l->depth);

if (\*list == NULL)

\*list = l;

else

line->next = l;

line = l;

if (l == objects.lines)

l = objects.lines = objects.lines->next;

else

l = ll->next = l->next;

line->next = NULL;

}

}

static void

sel\_spline(int xmin, int ymin, int xmax, int ymax) {

F\_spline\* s;

int urx, ury, llx, lly;

for (s = objects.splines; s != NULL; s = s->next)

{

if (!active\_layer(s->depth))

continue;

spline\_bound(s, &llx, &lly, &urx, &ury);

if (xmin > llx)

continue;

if (xmax < urx)

continue;

if (ymin > lly)

continue;

if (ymax < ury)

continue;

s->tagged = 1 - s->tagged;

toggle\_splinehighlight(s);

}

}

static void

get\_spline(F\_spline\*\* list)

{

F\_spline\* spline, \* s, \* ss;

for (s = objects.splines; s != NULL;)

{

//iterate until we find a selected spline

if (!s->tagged)

{

ss = s;

s = s->next;

continue;

}

remove\_depth(O\_SPLINE, s->depth);

if (\*list == NULL)

\*list = s;

else

spline->next = s;

spline = s;

if (s == objects.splines)

s = objects.splines = objects.splines->next;

else

s = ss->next = s->next;

spline->next = NULL;

}

}

static void

sel\_text(int xmin, int ymin, int xmax, int ymax) i

{

F\_text\* t;

int txmin, txmax, tymin, tymax; int dum;

for (t = objects.texts; t != NULL; t = t->next)

{

if (!active\_layer(t->depth))

continue;

text\_bound(t, &txmin, &tymin, &txmax, &tymax, &dum, &dum, &dum, &dum, &dum, &dum, &dum, &dum);

if (xmin > txmin || xmax < txmax || ymin > tymin || ymax < tymax)

continue;

t->tagged = 1 - t->tagged;

toggle\_texthighlight(t);

}

}

static void

get\_text(F\_text\*\* list)

{

F\_text\* text, \* t, \* tt;

for (t = objects.texts; t != NULL;)

{

//iterate until we find a selected text object

if (!t->tagged)

{

tt = t;

t = t->next;

continue;

}

remove\_depth(O\_TXT, t->depth);

if (\*list == NULL)

\*list = t;

else

text->next = t;

text = t;

if (t == objects.texts)

t = objects.texts = objects.texts->next;

else

t = tt->next = t->next;

text->next = NULL;

}

}

static void

sel\_compound(int xmin, int ymin, int xmax, int ymax) {

F\_compound\* c;

for (c = objects.compounds; c != NULL; c = c->next)

{

if (!any\_active\_in\_compound(c))

continue;

if (xmin > c->nwcorner.x)

continue;

if (xmax < c->secorner.x)

continue;

if (ymin > c->nwcorner.y)

continue;

if (ymax < c->secorner.y)

continue;

//tag compound as selected

c->tagged = 1 - c->tagged;

toggle\_compoundhighlight(c);

}

}

static void

get\_compound(F\_compound\*\* list)

{

F\_compound\* compd, \* c, \* cc;

for (c = objects.compounds; c != NULL;)

{

//iterate until we find a selected compound

if (!c->tagged)

{

cc = c;

c = c->next;

continue;

}

remove\_compound\_depth(c);

if (\*list == NULL)

\*list = c;

else

compd->next = c;

compd = c;

if (c == objects.compounds)

c = objects.compounds = objects.compounds->next;

else

c = cc->next = c->next;

compd->next = NULL;

}

}

//-----------------------------MOVE CODE-------------------------------------------------------------------------------- -----------

//---------------------------------------------------------------------------------- -----------------------------------------------

void

move\_selected1(void)

{

set\_mousefun("move object", "horiz/vert move", "", LOC\_OBJ, LOC\_OBJ, LOC\_OBJ); canvas\_kbd\_proc = null\_proc;

canvas\_locmove\_proc = null\_proc;

init\_searchproc\_left(init\_arb\_move);

init\_searchproc\_middle(init\_constrained\_move);

canvas\_leftbut\_proc = object\_search\_left;

canvas\_middlebut\_proc = object\_search\_middle;

canvas\_rightbut\_proc = null\_proc;

return\_proc = break\_comp;

set\_cursor(pick9\_cursor);

reset\_action\_on();

}

static void

init\_arb\_move(F\_line\* p, int type, int x, int y, int px, int py) {

constrained = MOVE\_ARB;

init\_move(p, type, x, y, px, py);

canvas\_middlebut\_proc = null\_proc;

set\_mousefun("place object", "", "cancel", LOC\_OBJ, LOC\_OBJ, LOC\_OBJ); draw\_mousefun\_canvas();

}

static void

init\_constrained\_move(F\_line\* p, int type, int x, int y, int px, int py) {

constrained = MOVE\_HORIZ\_VERT;

init\_move(p, type, x, y, px, py);

canvas\_middlebut\_proc = canvas\_leftbut\_proc;

canvas\_leftbut\_proc = null\_proc;

set\_mousefun("", "place object", "cancel", LOC\_OBJ, LOC\_OBJ, LOC\_OBJ); draw\_mousefun\_canvas();

}

static void

init\_move(F\_line\* p, int type, int x, int y, int px, int py) {

/\* turn off all markers \*/

update\_markers(0);

switch (type) {

case O\_COMPOUND:

set\_cursor(wait\_cursor);

cur\_c = (F\_compound\*)p;

list\_delete\_compound(&objects.compounds, cur\_c);

redisplay\_compound(cur\_c);

set\_cursor(null\_cursor);

init\_compounddragging(cur\_c, px, py);

break;

case O\_POLYLINE:

set\_cursor(wait\_cursor);

cur\_l = (F\_line\*)p;

list\_delete\_line(&objects.lines, cur\_l);

redisplay\_line(cur\_l);

set\_cursor(null\_cursor);

init\_linedragging(cur\_l, px, py);

break;

case O\_TXT:

set\_cursor(wait\_cursor);

cur\_t = (F\_text\*)p;

list\_delete\_text(&objects.texts, cur\_t);

redisplay\_text(cur\_t);

set\_cursor(null\_cursor);

init\_textdragging(cur\_t, x, y);

break;

case O\_ELLIPSE:

set\_cursor(wait\_cursor);

cur\_e = (F\_ellipse\*)p;

list\_delete\_ellipse(&objects.ellipses, cur\_e);

redisplay\_ellipse(cur\_e);

set\_cursor(null\_cursor);

init\_ellipsedragging(cur\_e, px, py);

break;

case O\_ARC:

set\_cursor(wait\_cursor);

cur\_a = (F\_arc\*)p;

list\_delete\_arc(&objects.arcs, cur\_a);

redisplay\_arc(cur\_a);

set\_cursor(null\_cursor);

init\_arcdragging(cur\_a, px, py);

break;

case O\_SPLINE:

set\_cursor(wait\_cursor);

cur\_s = (F\_spline\*)p;

list\_delete\_spline(&objects.splines, cur\_s);

redisplay\_spline(cur\_s);

set\_cursor(null\_cursor);

init\_splinedragging(cur\_s, px, py);

break;

default:

return;

}

}

//-------------------------------BREAK CODE--------------------------------------------------------------

//---------------------------------------------------------------------------------- --------------------

static void

init\_break(void)

{

cur\_c = temp;

mask\_toggle\_compoundmarker(cur\_c);

clean\_up();

list\_delete\_compound(&objects.compounds, cur\_c);

tail(&objects, &object\_tails);

append\_objects(&objects, cur\_c, &object\_tails);

toggle\_markers\_in\_compound(cur\_c);

set\_tags(cur\_c, 0);

set\_action(F\_BREAK); set\_latestcompound(cur\_c); set\_modifiedflag(); ffst\_selected();

}

**e\_ffst.h:**

#include "object.h"

#ifndef E\_FFST\_H

#define E\_FFST\_H

//Initials

void init\_border\_drawing(int x, int y);

void move\_selected1(void);

extern void ffst\_selected(void);

static void init\_break(void);

//Glue

static void create\_selectArea(int x, int y);

static void cancel\_box(void);

static void create\_compoundobject(int x, int y);

static int compose\_compound1(F\_compound \*c);

void tag\_obj\_in\_region1(int xmin, int ymin, int xmax, int ymax); static void sel\_line(int xmin, int ymin, int xmax, int ymax);

static void get\_line(F\_line\*\* list);

static void get\_arc(F\_arc\*\* list);

static void sel\_arc(int xmin, int ymin, int xmax, int ymax); static void get\_compound(F\_compound\*\* list);

static void sel\_compound(int xmin, int ymin, int xmax, int ymax); static void get\_ellipse(F\_ellipse\*\* list);

static void sel\_ellipse(int xmin, int ymin, int xmax, int ymax); static void sel\_line(int xmin, int ymin, int xmax, int ymax); static void get\_spline(F\_spline\*\* list);

static void sel\_spline(int xmin, int ymin, int xmax, int ymax); static void get\_text(F\_text\*\* list);

static void sel\_text(int xmin, int ymin, int xmax, int ymax); //Move

static void init\_move(F\_line\* p, int type, int x, int y, int px, int py), init\_arb\_move(F\_line\* p, int type, int x, int y, int px, int py), init\_constrained\_move(F\_line\* p, int type, int x, int y, int px, int py);

//This allows for the compound to be broken in in init\_break after it is moved F\_compound\* temp;

#endif

**mode.c:**

#define F\_PLACE\_LIB\_OBJ 17

+ #define F\_FFST 18

**w\_modepanel.c:**

… **includes**

#include "e\_deletept.h"

#include "e\_edit.h"

+ #include "e\_ffst.h"

#include "e\_glue.h"

#include "e\_joinsplit.h"

… **function declarations**

static void stub\_areameas\_selected(void);

static void stub\_tangent\_selected(void);

+ static void stub\_ffst\_selected(void);

… **in mode switches definition**

{&tangent\_ic, F\_TANGENT, tangent\_selected, M\_TANGENT\_OBJECT, I\_TANGENT,

"Add TANGENT/NORMAL to curve (n)",

False, NULL, (Pixmap)0, (Pixmap)0},

+ {&ffst\_ic, F\_FFST, ffst\_selected, M\_ALL, I\_OBJECT,

+ "Select Tool",

+ False, NULL, (Pixmap)0, (Pixmap)0},

… **at end of file**

static void

stub\_lenmeas\_selected(void)

{

change\_mode(&lenmeas\_ic);

}

+ static void

+ stub\_areameas\_selected(void)

+{

+ change\_mode(&ffst\_ic);

+}

**w\_icons.c:**

… **in “icons for mode panel” section**

/\*\*\*\*\*\*\*\*\*\*\* icons for mode panel \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

+ #define ffst\_width\_small 22

+ #define ffst\_height\_small 22

+ static char ffst\_bits\_small[] = {

+ 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

+ 0x80,0x17,0x00,0x20,0x20,0x00,0x10,0x40,0x00,0x00,0x80,0x00,

+ 0x08,0x00,0x01,0x08,0x00,0x01,0x08,0x00,0x00,0x08,0x00,0x01,

+ 0x08,0x00,0x01,0x08,0x00,0x00,0x10,0x80,0x00,0x00,0x00,0x00,

+ 0x60,0x20,0x00,0x80,0x0d,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

+ 0x00,0x00,0x00,0x00,0x00,0x00};

… **in “NORMAL SIZE ICONS” section**

/\*\*\*\*\*\* NORMAL SIZE ICONS \*\*\*\*\*\*/

+ #define ffst\_width\_big 36

+ #define ffst\_height\_big 32

+ static unsigned char ffst\_bits\_big[] = {

+ 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

+ 0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x00,0x00,0x00,0x04,

+ 0x00,0x01,0x00,0x00,0x04,0x00,0x01,0x00,0x00,0x02,0x00,

+ 0x02,0x00,0x00,0x01,0x00,0x04,0x00,0x00,0x01,0x00,0x04,

+ 0x00,0x80,0x00,0x00,0x08,0x00,0x80,0x00,0x00,0x08,0x00,

+ 0x40,0x10,0x00,0x10,0x00,0x20,0x18,0x00,0x20,0x00,0x20,

+ 0x10,0x07,0x20,0x00,0x10,0x10,0x07,0x40,0x00,0x20,0x10,

+ 0x07,0x20,0x00,0x20,0x10,0x00,0x20,0x00,0x40,0x38,0x00,

+ 0x10,0x00,0x80,0x00,0x00,0x08,0x00,0x80,0x00,0x00,0x08,

+ 0x00,0x00,0x01,0x00,0x04,0x00,0x00,0x01,0x00,0x04,0x00,

+ 0x00,0x02,0x00,0x62,0x00,0x00,0x04,0x00,0x91,0x00,0x00,

+ 0x04,0xc0,0x81,0x00,0x00,0xf8,0xff,0x41,0x00,0x00,0x00,

+ 0xc0,0x21,0x00,0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x00,

+ 0xf0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

+ 0x00,0x00,0x00,0x00,0x00,0x00};

… **in struct definition section**

icon\_struct lenmeas\_ic;

icon\_struct areameas\_ic;

+ icon\_struct ffst\_ic;

… **populate\_icons\_big()**

icon\_struct areameas\_ic\_ = { areameas\_width\_big, areameas\_height\_big, (char\*)areameas\_bits\_big };

+ icon\_struct ffst\_ic\_ = { ffst\_width\_big, ffst\_height\_big, (char\*)ffst\_bits\_big };

+ ffst\_ic = ffst\_ic\_;

regpoly\_ic = regpoly\_ic\_;

… **populate\_icons\_small()**

icon\_struct areameas\_ic\_ = { areameas\_width\_small, areameas\_height\_small, (char\*)areameas\_bits\_small };

+ icon\_struct ffst\_ic\_ = { ffst\_width\_small, ffst\_height\_small, (char\*)ffst\_bits\_small };

+ ffst\_ic = ffst\_ic\_;

regpoly\_ic = regpoly\_ic\_;

**w\_icons.h:**

extern icon\_struct lenmeas\_ic;

extern icon\_struct areameas\_ic;

+ extern icon\_struct ffst\_ic;