SOFTWARE TOOLS FOR THE GRAPHIC DESIGNER

by
Nardy Henigan

B.F.A., Northern Illinois University
DeKalb, Illinois
June, 1977
M.A., Northern Illinois University
DeKalb, Illinois
August, 1978

Submitted to the Department of Architecture
in partial fulfillment of the requirement of the
Degree of Master of Science in Visual Studies
at the Massachusetts Institute of Technology

June, 1983

Signature of Author _____
Nardy Henigan
Department of Architecture, May 5, 1983

Certified by _____
Muriel Cooper
Associate Professor of Visual Studies
Thesis Supervisor

Accepted by _____
Nicholas Negroponte, Chair
Departmental Committee for
Graduate Students

SOFTWARE TOOLS FOR THE GRAPHIC DESIGNER

by
Nardy Henigan

Submitted to the Department of Archtecture on May 5, 1983, in partial fulfillment of the requirements for the Degree of Master of Science in Visual Studies

Abstract

This thesis describes three packages of programs of interest to the graphic designers: (1) a tool for pointing at objects on the display monitor, (2) a tool for organizing sets of programs into interactive menus, and (3) a tool for digitizing type fonts.

The graphic designer at the computer uses at least three spaces at once: the page he is designing, the device space of the display monitor, and the device space of the graphics tablet. The first tool figures the transforms necessary to map one space into another. This allows the user to point at a location on the tablet and thereby identify corresponding points on the screen and in user-defined space.

Interactive page layout requires the orchestration of large numbers of separate procedures. Choosing between these procedures can be made easier for the user if he is presented with a menu of options when a decision is required. The second tool is a package of programs to help the programmer create and manage simple or complex menus.

Simulating type on a monitor requires digitized fonts. The third tool is a package of programs which extracts the edge of a letterform image in the frame buffer and stores it as a chain code.

Thesis supervisor: Muriel Cooper
Title: Associate Professor of Visual Studies

CONTENTS

1 INTRODUCTION

1.1 SOFTWARE TOOLS

1.1.1 "Programmes for solutions"

The arrival of the computer at the graphic designer's workstation -- or the arrival of the graphic designer at the computer workstation -- recalls a statement by Karl Gerstner, a Swiss graphic designer, who advocated "instead of solutions for problems, programmes for solutions." [1] He proposed that graphic designers focus their efforts on the design process rather than directly on final solutions. Gerstner's "programmes" were not computer programs, but were general methods for approaching visual problems, and systematically generating alternative solutions. However he anticipated the role of a growing number of graphic designers who need to express their methodology as computer code.

1.1.2 Presummary

This thesis describes three packages of programs of interest to the graphic designers: (1) a tool for

communicating with the computer by pointing, (2) a tool for organizing sets of programs into interactive menus, and (3) a tool for digitizing type fonts.

1.1.2.1 The graphic designer at the computer uses at least three different spaces: the space of the page he is designing, the device space of the display monitor, and the device space of the graphics tablet. The first tool figures the transformations necessary to map one space into another. This allows the user to point at a location on the tablet and thereby identify corresponding points on the screen and in the user-defined space.

1.1.2.2 Interactive page layout requires the orchestration of large numbers of separate procedures. Choosing between these procedures can be made easier for the user if he is presented with a menu of options when a decision is required. The second tool is a package of programs to help the programmer create and manage simple or complex menus.

1.1.2.3 Simulating type on a monitor requires digitized fonts. The third tool is a package of programs which extracts the edge of a letterform image in the frame buffer and stores it as a chain code.

1.1.3 Method of explication

Each software tool is presented and developed according to the following plan.

1.1.3.1 Problem description. The presentation of each software tool begins with the description of a problem of interest to graphic designer/programmers. I explain why this is a significant problem, what are the constituent parts of the problem, and what are the current ways of treating it both in software and hardware. I discuss the general concepts, formulas and algorithms involved in the solution, and compare and contrast my solution with alternative methods.

1.1.3.2 Program documentation. A working implementation of the software tool is documented. A users manual is presented for a package of pl/1 programs on the MagicSix operating system at the Visible Language Workshop. Complete debugged pl/1 source code listings are included in the appendix.

1.1.3.3 Evaluation and further work. I discuss the currently implemented programs with respect to completeness, robustness, consistency, and other observed weaknesses and strengths. I discuss ways of increasing speed of execution and reducing storage, since space and time are always at a premium. In addition, I suggest ways of making the code more more machine-, system-, and language-independent. This makes the packages easier to fix and maintain, accomodates growth and change, and makes it easier to move the packages to other computers.

1.1.4 Computing environment

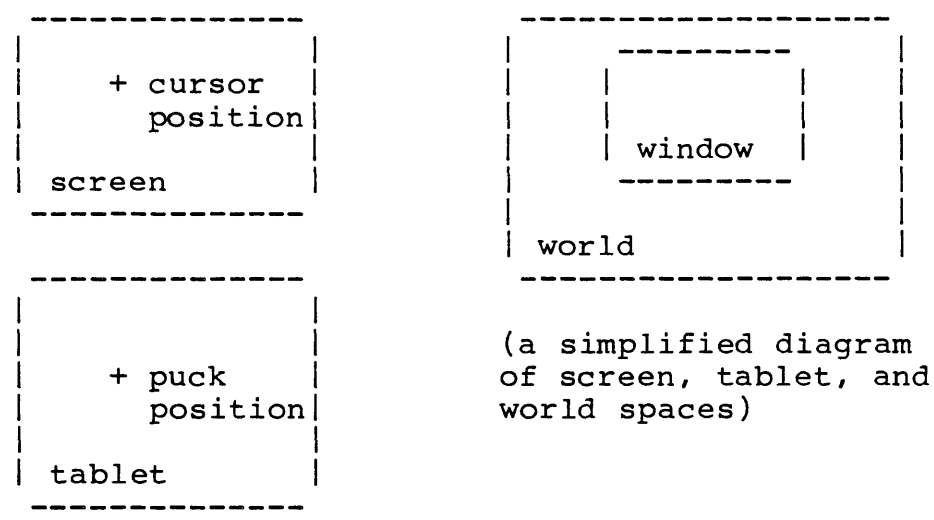
1.1.4.1 Hardware. At the outset a general overview of the system will be helpful. The Perkin-Elmer 3220 is a 32-bit minicomputer with 512K of core memory. Disk storage is a 300 megabyte "trident-type" drive with a high I/O bandwidth. Peripherals include: a Grinnell GMR-270 frame buffer with a capacity of 512 x 512 pixels with 27 bits of color information, a Vidicon surveillance camera which feeds non-composite video signals to the frame buffer, and a Summagraphics Bit Pad tablet with a 4-button puck.

1.1.4.2 Software. The MagicSix operating system, developed by the Architecture Machine Group at M.I.T., supports an interactive computing environment. Features include a tree-structured file system, dynamic linking, and a process stack. The pl/1 language, a subset of standard PL/1, was also developed by the Architecture Machine Group. It supports recursion, pointers, structures, signals, error-handling, and "initiated segments," which allow the user to structure core memory.

2 POINTING AND MAPPING

2.1 THE IDEAS

2.1.1 World and device space



(a simplified diagram of screen, tablet, and world spaces)

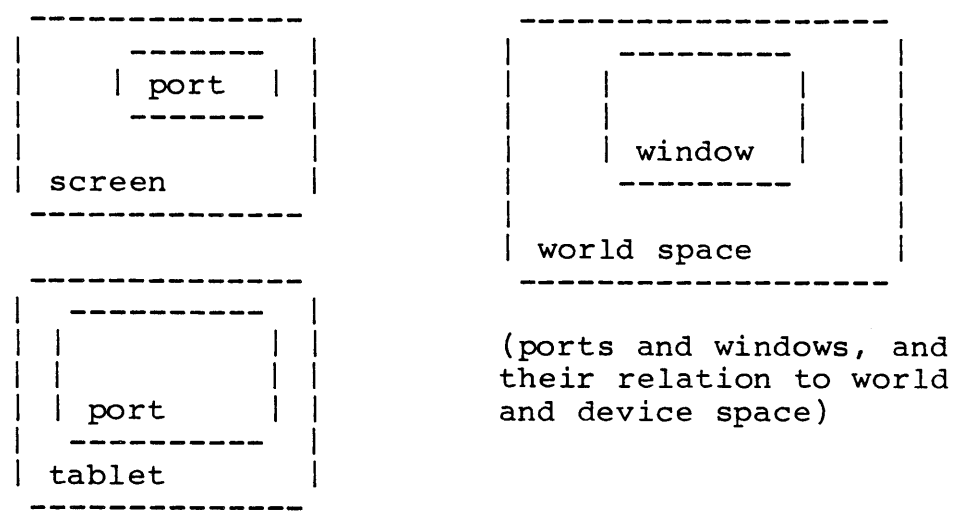
2.1.1.1 The designer of graphics programs deals with at least three different kinds of spaces. (1) A "window" of interest in "world space:" some examples are an 8 1/2 x 11 page, a two-page spread, a business card, a box car, an integer grid, or a floating point slider. (2) A display screen: composed of discrete pixels, numbered as positive integers in the first quadrant of a Cartesian coordinate system. The screen is used to model world space. (3) A graphics tablet: which senses the position of a puck on a

grid of charged wires and which returns positive (x,y) integers. The tablet is used to model of both world and screen space.

2.1.1.2 As the puck is moved about on the tablet, the corresponding screen position can be determined with a mapping transform, and a mark such as a cursor can be displayed nondestructively at that point. The user can in this manner "point" at images on the screen and at objects in world space.

2.1.1.3 In some applications, such as "painting" and picture-making systems, the world window is identical to the display screen. But in the case of page layout, the window is the page being designed, and neither is it the size of the screen, nor do page units (e.g., points and picas) align with pixel boundries.

2.1.2 Ports and windows



(ports and windows, and their relation to world and device space)

2.1.2.1 The window is a subset of all points in world space, and the user may likewise establish a "port" on the screen or tablet which is a subset of all the addressable points in the device space. The reader should note that the term "window" is used in reference to world space, and that "port" is used in reference to device space. This is standard computer graphics usage. [2]

2.1.1.2 To map a point from a window to a port, the following formulas are used:

```
port_x = (window_x - window_x_origin) *
          (port_x_extent / window_x_extent)
port_y = (window_y - window_y_origin) *
          (port_y_extent / window_y_extent)
```

2.1.2.3 Equations of the same general form can be used for the following mappings:

world window to screen port: answers the question "where is a point in the world located on the screen?"

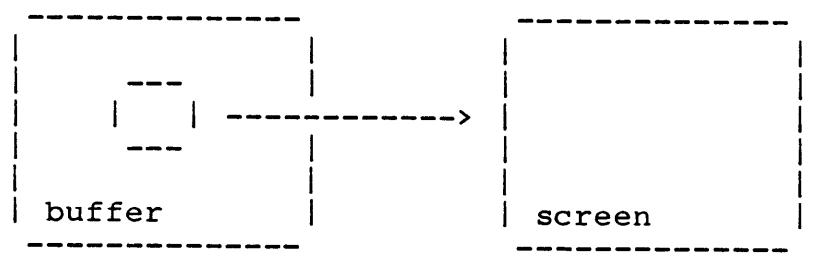
tablet port to world window: answers the question "where is the user pointing in the world when he locates the puck at a point on the tablet?"

tablet port to screen port: answers the question "where is the user pointing at on the screen when he locates the puck at a point on the tablet?"

2.1.3 Zoom facility

2.1.3.1 The discussion is now expanded to distinguish between the screen, an grid of glowing phospher points, and the frame buffer, an matrix of memory cells where the image information is stored. There is not always a 1-1 mapping

from the buffer to the frame screen array because some buffers allow a subset of memory cells to be scanned and displayed at a larger size on the screen.



(zoom example: 1/16 of the buffer is displayed on the screen at 4 times normal size)

2.1.3.2 This feature is called zoom. The Grinnell GMR-27Ø buffer, for example, can display a full image at full scale, or zoom to 1/4 image at 2 times scale, 1/16 at 4, and 1/64 at 8. Other buffers allow zooming to all integral values between 1 and 8, and still others allow continuous zooming from 1.Ø to 8. Ø (and beyond). The zoom capacity allows closer inspection of the image, and can also allow greater precision in mapping tablet locations to the buffer locations -- both desirable capabilities.

2.1.3.3 The Grinnell extracts a price for this facility, however. The cursor is not stored in the frame buffer, but is added to the rgb signal after scanning buffer memory. The cursor, therefore, does not alter any part of the stored image, but is displayed via hardwired circuits that do not adjust the position or size of the cursor to account for zoom. The map package described in section 2.2 corrects the cursor position so it is in register with the zoomed pixel it points at.

2.1.3.4 Another consideration is that when zoom scale is greater than 1 some or even all of the port (into which the world window and tablet port are mapped), may fall outside of the displayed section of the buffer.

2.1.3.5 A final consideration is that it may be desirable or convenient to have alternate ways to to specify which part of the buffer to zoom to. Three different ways would probably handle all needs:

1. specify scale and point to be at origin of screen when buffer is displayed.

2. specify scale and point to be in center of screen when buffer is displayed.

3. specify scale and an invariant point, around which all scaling occurs, and which would not change screen position before and after scaling.

2.2 THE MAP PACKAGE: A USER MANUAL

2.2.1 Introduction to map$ package

2.2.1.1 This set of routines facilitates the user's control of three two-dimensional spaces: the screen, the tablet, and the world space of interest to the user. The procedures of this package are bound together as a single segment named "map". A structure containing parameters accessed by all procedures is in a separate segment named "map_param_Ø1" which is managed automatically by the package.

2.2.1.2 Use of space. The user may have as many virtual core memory spaces as he can think up four-letter names for. Each space is 16 (addressed 'Ø'b4 to 'f'b4) 32K-byte (addressed 'ØØØØ'b4 to '7fff'b4) segments. Each segment is either object code or data. The following schedule shows a likely distribution of segments in a space when the map package is used:

```
address   reference name
     Ø    [unavailable to user]
     1    [unavailable to user]
     2    ioa, iocs, rl, scs, stty
     3    grin
     4    math
     5    use [the menu manager program]
     6    map
     7    io_util
     8    com_err
     9    [user application procedure]
     a    [user data segment]
     b    map_param_Ø1
     c    [menu tree data structure]
     d    [unavailable to user]
     e    linkage [and storage for based]
     f    [storage for automatic variables]
```

Each map procedure refers to and stores data in map_param_Ø1. The pointer to this segment is not passed as a parameter, but is initialized automatically to the same address (b|ØØØØ) at the start of each procedure.

2.2.1.3 %include. This file, when included in source code of an application program, declares all map$ procedures.

2.2.2 The map$ procedures

2.2.2.1 map$setup. This procedure is declared and used as follows:

```
declare map$setup entry;
call map$setup;
```

The procedure must be called before any other map$procedure. It calls map$load_parameters, which appends (if necessary) the "map_param_Ø1" data segment in the current working directory and initiates it at address 'b'b4 in user space (see map$load_parameters). It also sets default values for each window:

coordinate space:	x_origin	y_origin	x_extent	y_extent
world window	Ø	Ø	512	512
screen/buffer port	Ø	Ø	512	512
tablet port	76	6ØØ	2Ø48	1536

The world window is identified with the screen/buffer port. Tablet port is based on four considerations: (1) most resolution possible in each direction, (2) comfortable fit of puck on tablet, (4) ratio of tablet device units/ screen pixels is an integral 3:1 for x, 4:1 for y, (4) more units in direction of greater arm movement. The zoom scale is set to 1, and cursor is turned on.

2.2.2.2 map$window. This procedure and associated variables are declared and used as follows:

```
declare
     map$window     entry (float(23), float(23),
                    float(23), float(23)),
     x_origin       float(23),
     y_origin       float(23),
     x_extent       float(23),
     y_extent       float(23);
call map$window
     (x_origin, y_origin, x_extent, y_extent);
```

The arguments are the origin and extent of the world window, expressed in world units. They define the range of values to be returned by the map$int and map$fp routines. Default parameters set by map$setup are (Ø.Ø, Ø.Ø, 511.Ø, 511.Ø);

2.2.2.3 map$port. This procedure and associated variables are declared and used as follows:

```
declare
     map$port        entry (fix(15), fix(15), fix(15),
                     fix(15)),
     x_origin        fix(15),
     y_origin        fix(15),
     x_extent        fix(15),
     y_extent        fix(15);
call map$port
     (x_origin, y_origin, x_extent, y_extent);
```

The arguments are the origin and extent of the frame buffer port. Values for the origin may range from Ø to 511. Origin + extent should not exceed 511 in either direction. At zoom scale 1, the frame buffer and screen image are identical. The arguments define the range of values to be returned by the map$int and map$fp routines. Default parameters set by map$setup are (Ø, Ø, 511, 511).

2.2.2.4 map$tablet. This procedure and associated variables are declared and used as follows:

```
declare
     map$tablet      entry (fix(15), fix(15), fix(15),
                     fix(15)),
     x_origin        fix(15),
     y_origin        fix(15),
     x_extent        fix(15),
     y_extent        fix(15);
call map$tablet
     (x_origin, y_origin, x_extent, y_extent);
```

The arguments are the origin and extent of the tablet port. Values for the origin may range from 0 to 2200. Origin + extent should not exceed 2200 in either direction. Default parameters set by map$setup are (76, 600, 2000, 1500);

2.2.2.5 map$x_port. This function and associated arguments are declared and used as follows:

```
declare
     map$x_port      entry (float(23))
                     returns (fix(15)),
     window_x        float(23),
     port_x          fix(15);
port_x = map$x_port (window_x);
```

This function transforms points in world space to buffer space. It does not check, clamp, or clip. So a point which is not inside the window will transform into a point which is not inside the port.

2.2.2.6 map$y_port. This function and associated arguments are declared and used as follows:

```
declare
     map$y_port      entry (float(23))
                     returns (fix(15)),
     window_y        float(23),
     port_y          fix(15);
port_y = map$y_port (window_y);
```

This function transforms points in world space to buffer space. It does not check, clamp, or clip. So a point which is not inside the window will transform into a point which is not inside the port.

2.2.2.7 map$fp. This procedure and associated arguments are declared as follows:

```
declare
     map$fp          entry (fix(15), fix(15),
                     float(23), float(23), fix(15)),
     port_x          fix(15),
     port_y          fix(15),
     fp_x            float(23),
     fp_y            float(23),
     z               fix(15);
call map$fp (port_x, port_y, fp_x, fp_y, z);
```

This procedure interrogates the tablet and maps the resulting tablet coordinates into the user window (port_x, port_y) and screen port (fp_x, fp_y). Under default conditions, cursor number one is displayed at (port_x, port_y). Cursor visibility and number (one through four)

can be controlled by calling map$cursor procedures.

z takes on the following values:

-1	puck is not on tablet
Ø	puck is on tablet, no buttons are pressed
1	puck is on tablet, z button is pressed (yellow)
2	puck is on tablet, button 1 is pressed (white)
4	puck is on tablet, button 2 is pressed (blue)
8	puck is on tablet, button 3 is pressed (green)
3-15	combination of buttons is being pressed

2.2.2.8 map$int. This procedure and associated arguments are declared as follows:

```
declare
     map$int          entry (fix(15), fix(15), fix(15),
                      fix(15)),
     port_x           fix(15),
     port_y           fix(15),
     int_x            fix(15),
     int_y            fix(15);
call map$int (port_x, port_y, int_x, int_y);
```

This procedure works analogously to map$int. int_x and int_y are derived by mapping the tablet location to the window, and truncating the floating point values to integers. port_x and port_y (and consequently the cursor position) are derived by mapping the integer window values to the port. This is useful, among other things, for grid gravity, or centering cursor on menu choices.

2.2.2.9 map$frame_org. This procedure and associated arguments are declared as follows:

```
declare
     frame_org      entry (fix(15), fix(15), fix(15)),
     x_origin       fix(15),
     y_origin       fix(15),
     scale          fix(15);
call map$frame_org (x_origin, y_origin, scale);
```

This procedure controls which part of the frame buffer is displayed. (x_origin y_origin) is the the pixel displayed in the lower left corner of the screen. Scale works this way for all of the map$frame procedures:

scale	pixels displayed
1	512 x 512
2	256 x 256
4	128 x 128
8	64 x 64

2.2.2.1Ø map$frame_ctr. This procedure and associated arguments are declared as follows:

```
declare
     frame_ctr      entry (fix(15), fix(15), fix(15)),
     x_center       fix(15),
     y_center       fix(15),
     scale          fix(15);
call map$frame_ctr (x_center, y_center, scale);
```

This procedure controls which part of the frame buffer is displayed. (x_center y_center) is the pixel displayed in the center of the screen. Scale works as described earlier.

2.2.2.11 map$frame_loc. This procedure and associated arguments are declared as follows:

```
declare
     frame_loc      entry (fix(15), fix(15), fix(15)),
     x_locus        fix(15),
     y_locus        fix(15),
     scale          fix(15);
call map$frame_loc (x_locus, y_locus, scale);
```

This procedure controls which part of the frame buffer is displayed. (x_locus y_locus) is an invariant pixel that does not change its displayed position as a result of calling this procedure.

2.2.2.12 map$frame_reset. This procedure is declared and called as follows:

```
declare map$frame_reset entry;
call map$frame_reset;
```

This is equivalent to calling map$frame_org (Ø, Ø, 1);

2.2.2.13 map$cursor. This procedure is declared and called as follows:

```
declare
     map$cursor     entry (fix(15), fix(15), fix(15)),
     number         fix(15),
     screen_x       fix(15),
     screen_y       fix(15);
call map$cursor (number, screen_x, screen_y);
```

This locates one of four cursors (1, 2, 3, or 4) at (screen_x, screen_y). Whether or not it is visible is controlled by map$cursor_vis. Scale (specified in map$frame calls) has no effect on cursor. It always maps to the screen as if scale were 1

2.2.2.14 map$cursor_nbr. This procedure is declared and called as follows:

```
declare
     map$cursor_nbr entry (fix(15)),
     number         fix(15);
call map$cursor_nbr (number);
```

This procedure controls which of four cursors (1, 2, 3, 4) is used when map$int or map$fp is called.

2.2.2.15 map$cursor_vis. This procedure is declared and called as follows:

```
declare
     map$cursor_vis entry (fix(15)),
     string4        bit(4);
call map$cursor_vis (string);
```

The procedure controls which of the four cursors is visible. The cursors 1, 2, 3, 4 correspond with string positions 1, 2, 3, 4. '0'b is off, '1'b is on. For example '1010'b turns cursors 1 and 3 on.

2.2.2.16 map$load_parameters. This procedure is declared and called as follows:

```
declare map$load_parameters entry;
call map$load_parameters;
```

This procedure creates (if necessary) and loads the data segment without initializing it. It is safe to call this procedure instead of map$setup only when the user is sure that "map_param_Ø1" has already been created and initialized by a previous program.

2.3 FURTHER WORK

2.3.1 Device-independent graphics package

An obvious improvement in this package is making it a part of a graphics package by adding line_to's, moveto's and clipping. A further step is making it more machine independent by changing the frame buffer space to normalized origin of (0.0, 0.0) and extent of (1.0, 1.0). It would then be adapted to each specific buffer by adding one procedure which would transform normalized buffer coordinates to actual device coordinates. This would achieve the usual tradeoff of greater generality for somewhat slower execution time.

2.3.2 Record and playback

Another extension would be an ability to store input from the tablet, and then at a later time read that data from memory as if it were coming from the tablet. Input from the keyboard could be stored and recalled in the same way.

This could be used for animations, and to record and replay interactive sessions. A simpler package to do this has already been written by the author, and saves about 45 seconds worth of data from the tablet before memory is filled.

2.3.3 System independence

These procedures communicate with each other by saving and recalling data in a separate segment. It is not clear that this could be accomplished as easily in a system without the explicit core memory management provided by the MagicSix system.

3 MENUS

3.1 THE IDEAS

3.1.1 Requirements for interactive graphics

3.1.1.1 An interactive package which accomplishes a task such as page layout requires a complex set of procedures. For the user to move from one procedure to the next in a purposeful, efficient way, requires an interface with the following characteristics:

3.1.1.2 The system must present options to the user and respond to his choice in a consistent manner. Predictable behavior makes it easier for the user to adjust to the system. The system should accommodate both new and experienced users, especially with respect to prompts, so that they are available to new users or can be ignored or bypassed by experienced users. The system should provide feedback to reinforce choices, and not allow unanticipated choices to crash the process. Also, some accomodation has to be made for the fact that different users use tools in different ways.

3.1.1.3 All this places special demands on the programmer who, in turn, has needs of his own to be met with respect to developing and coordinating large sets of programs. He needs some way of organizing the procedures so that only the ones of immediate interest are presented to the user to select. He needs some way of managing core memory so that procedures don't hang around when they are no longer needed (some operating systems take care of this for the programmer). The changeability, modularity, and communication of values between procedures.

3.1.2 Definition of a menu

3.1.2.1 The intuitive answer seems to call for some form of a menu, but this only seems to presents further questions: what is a menu? Is it a procedure or data? Do menus run programs, or do programs run menus?

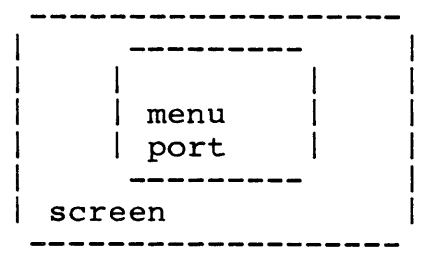
3.1.2.2 I will develop some answers to these questions based on a recursive definition of a menu: a menu is a list of procedure-items and menu-items such that if the user selects a procedure-item a procedure is run, and if the user selects a menu-item another menu is presented.

3.1.2.3 This is vague enough to postpone some implementation considerations for a while, but it still permits some useful comparisons. The structure of a menu like this can be repesented by a tree with programs at its leaves, or terminal nodes, and menus as its root and intermediate nodes. Another analog is an outline. Any item with items indented beneath it represents a menu, and any item without items indented beneath it represents a program. The outline is a fortuitous analogy, as we shall see, because it is a familiar form, it is easy to write and edit and manipulate.

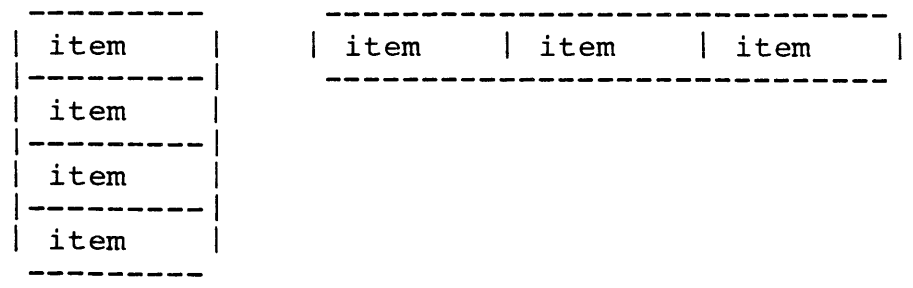
3.1.2.4 Both the tree and the outline can be modelled by dynamic information structures known as hierarchical linked lists. Since there are well-defined techniques for building, editing, and traversing linked lists [3], it is now possible to postulate a menu that is a data structure which is created and managed by special programs.

3.1.3 Visual format

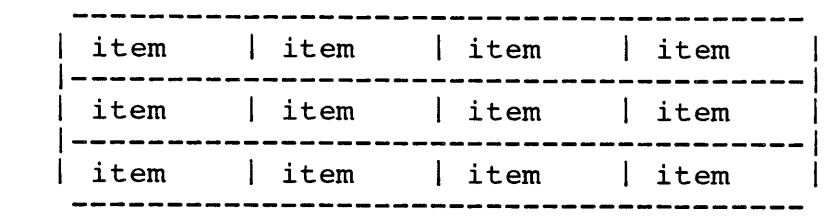
3.1.3.1 So far I have described the menu in a non-visual way. The visual form can be derived by describing general its general properties. A menu is a list of items displayed in a screen port. It should be able to occupy any subarea of the screen.



3.1.3.2 A menu port may contain vertical or horizontal lists. To this end, the port may be divided into equal-sized modules in which menu items are displayed.

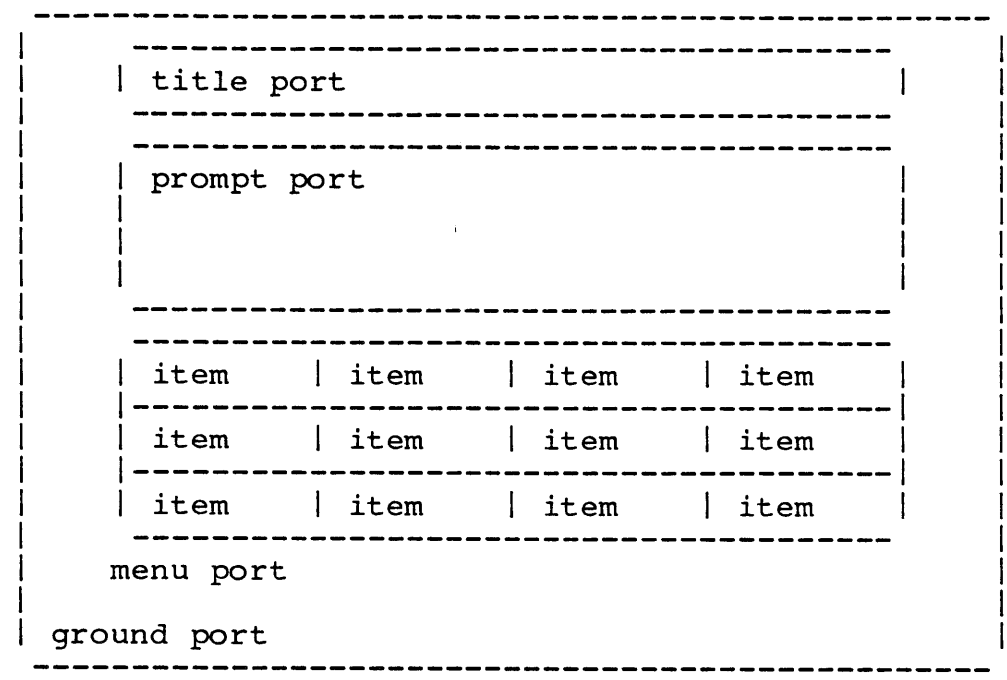


3.1.3.3 Geometry suggests that a port may contain arrays in addition to simple lists of these modules. This is, in fact, useful for several purposes. A single list might be displayed more compactly as an array. Or this format might display vertical lists of horizontal lists (and horizontal lists of vertical lists) -- menu trees, such as I earlier described.



3.1.3.4 One of the advantages that accrues from dividing the menu port into equal modules is that the position of the puck on the tablet can be mapped easily and unambiguously to the location of a menu item.

3.1.3.5 In addition to the item modules, a menu should include prompts and feedback so that a mimimum configuration might include a menu port, a title port, and a prompt port all within a containing "ground port."



(diagram of parts of a menu)

3.2 THE MENU PACKAGE: A USER MANUAL

3.2.1 The menu text file

3.2.1.1 This package of procedures bound together under the name "menu" turns a text file into a data structure -- a menu tree -- which is used for interactive decision-making. A model menu is included in Appendix III. The following paragraphs describe the four main sections (and functions) of the text file.

3.2.1.2 The text file is created and edited in the TVmacs editor. The text file name should have the suffix ". menu", just as pll programs have the suffix ".pll".

3.2.1.3 The user may insert a comment on any line by typing a vertical bar (|). All characters from the vertical bar to the end of the line will be ignored, and do not become part of the menu data structure.

3.2.2 The menu outline

3.2.2.1 The programmer writes an outline that reflects the pattern of choices he wants to make available on the screen. The outline is made of menu-items and procedure-items. Menu-items are followed on the same line by prompts, which give information about items indented one further level beneath them (if the user picks a menu item, another menu appears on the screen). Procedure-items are followed on the same line by a procedure which is run if that item is chosen.

3.2.2.2 These are the rules for making a menu outline. The characters that flag the start of this section are "**MENU" (no quotation marks) at the beginning of a line. There is one item per line. There is only 1 first-level item: the name of the menu (no indent, flush left) followed by the first-level prompt. To indent one from the preceeding item, insert 1 extra tab, or 5 spaces, before the next label. No numbers or letters (I., A., 1., a.) are necesary, use only indents.

3.2.2.3 The syntax for a menu-item that results in another menu is:

```
label "prompt"
label [match string]
```

If the prompt will fit on one line following the label, single or double quotes may be used. Otherwise, put a match string in brackets after the label, and put the same match string in brackets in the prompts section, followed by the full text of the actual prompt.

3.2.2.4 The syntax for an item that calls a procedure is:

```
label: procedure_name
```

For readability, extra carriage returns can be inserted between lines. Extra tabs and spaces can be included between required elements. Tabs or spaces are significant only at the beginning of the line.

3.2.2.5 Keep the labels short. Any characters which don't fit in the boxes (see the ports section) are truncated, and not displayed.

3.2.3 Prompts

3.2.3.1 Prompts give the user information on items at the next sublevel. Because the outline is restricted to one line per item, the prompts section allows the programmer to write as detailed a prompt as is needed, and to associate it with a label by a match string in brackets.

3.2.3.2 The characters that flag the start of this section are "**PROMPTS" (no quotation marks). The syntax for each prompt is as follows:

```
[match string]<cr>
This is whatever expanded prompt is wanted.
It may use as many lines as necessary.
The match string should be on a line by itself.
No quotation marks are necessary.
```

in the above example everything that follows the first carriage return <cr> is the full text of the prompt (and in the file is flush left, not indented).

3.2.3.3 Currently, you need to include at least one prompt in this section whether you really need it or not. This is an inconvenience that will be fixed.

3.2.4 Ports

3.2.4.1 The characters that flag the start of this section are "**PORTS" (no quotation marks) at the beginning of a line.

3.2.4.2 The programmer specifies the visual format numerically by describing the dimensions of different parts of the menu. Specification syntax is: keyword followed by 10 parameters, separated by blanks or tabs, all on a single line. The parts of the menu, or ports, are as follows:

3.2.4.3 Keyword "ground"

```
(this port contains the whole menu)
x_origin        relative to screen origin
y_origin        relative to screen origin
width           in pixels
height          in pixels
x_char_pos      not relevant (put dummy parameter here)
y_char_pos      not relevant (put dummy parameter here)
char_width      not relevant (put dummy parameter here)
char_height     not relevant (put dummy parameter here)
outline_width   width, in pixels, of the box outline
planes_810      3-character string xxx where "x" is
                0 if plane is not used, 1 if used
```

3.2.4.4 Keyword "menu"

(this port contains all the menu item boxes)

x_origin	relative to ground origin
y_origin	relative to ground origin
width	in pixels
height	in pixels
x_char_pos	not relevant (put dummy parameter here)
y_char_pos	not relevant (put dummy parameter here)
char_width	not relevant (put dummy parameter here)
char_height	not relevant (put dummy parameter here)
outline_width	width, in pixels, of the box outline
planes_891Ø	3-character string xxx where "x" is Ø if plane is not used, 1 if used

3.2.4.5 Keyword "item"

(these ports contain the menu labels)

x_origin	not relevant (put dummy parameter here)
y_origin	not relevant (put dummy parameter here)

[x_origin and y_origin are taken to be (Ø, Ø)]

width	in pixels
height	in pixels
x_char_pos	first char, relative to title origin
y_char_pos	first char, relative to title origin
char_width	character spacing, in pixels
char_height	linespacing, in pixels
outline_width	width, in pixels, of the box outline
planes_891Ø	3-character string xxx where "x" is Ø if plane is not used, 1 if used

3.2.4.6 Keyword "title"

(this port contains the label of item which is currently selected)

x_origin	relative to ground origin
y_origin	relative to ground origin
width	in pixels
height	in pixels
x_char_pos	first char, relative to title origin
y_char_pos	first char, relative to title origin
char_width	character spacing, in pixels
char_height	linespacing, in pixels
outline_width	width, in pixels, of the box outline
planes_891Ø	3-character string xxx where "x" is Ø if plane is not used, 1 if used

3.2.4.7 Keyword "prompt"

(this port contains all the prompts)	
x_origin	relative to ground origin
y_origin	relative to ground origin
width	in pixels
height	in pixels
x_char_pos	first char, relative to prompt origin
y_char_pos	first char, relative to prompt origin
char_width	character spacing, in pixels
char_height	linespacing, in pixels
outline_width	width, in pixels, of the box outline
planes_891Ø	3-character string xxx where "x" is Ø if plane is not used, 1 if used

3.2.5 Options

3.2.5.1 The characters that flag the start of this section are "**OPTIONS" (no quotation marks). at the beginning of a line. The options are specified as follows:

3.2.5.2 Keyword "leaf_prefix" followed by "yes" or "no". If "yes" an asterisk is put before labels in procedure-item boxes. Otherwise not.

3.2.5.3 Keyword "plane_8" followed by "black", "red", "blue", "green", "cyan", "magenta", "yellow", or "white". This option colorizes overlay plane 8.

3.2.5.4 Keyword "plane_9" followed by same possible arguments as plane_8, with corresponding effect.

3.2.5.5 Keyword "plane_1Ø" followed by same possible arguments as plane_9, with corresponding effect.

3.2.5.6 Keyword "mode" followed by "pop_up" or "continuous". In continuous mode, the menu is displayed all the time. In pop_up mode, it goes away after user has chosen a procedure and reappears after the procedure has finished executing.

3.2.6 Making the menu

After the text file has been completed, it is turned into a menu-tree data file by running the menu program with the text file name (with or without the ".menu") as an argument:

```
menu textfile
or
menu textfile.menu
```

It is analagous to compiling a pll program. The menu program does not print error messages, but does print out

the current line(s) in the text file that it is working on. If the menu program finds some syntax error it can't handle, the user then should then inspect that line of the textfile. After the menu-tree file is made, it can be used by the "use" program.

3.3 THE USE PACKAGE: A USER MANUAL

3.3.1 Using the menu data segment

The menu program processes a text file called "something.menu" into a data file which it calls simply "something". Using the menu requires the user to type "use something" at command level. If he types simply "use", the use program will prompt for the menu name.

3.3.2 Use of space

As noted in the map section, the user may have as many virtual core memory spaces as he can think up four-letter names for. Each space is 16 (addressed 'Ø'b4 to 'f'b4) 32K-byte (addressed 'ØØØØ'b4 to '7fff'b4) segments. Each segment is either object code or data. The following schedule shows a likely distribution of segments in a space when the use package is used (note that "use" calls "map"):

```
address   reference name
     Ø    [unavailable to user]
     1    [unavailable to user]
     2    ioa, iocs, rl, scs, stty
     3    grin
     4    math
     5    use [the menu manager program]
     6    map
     7    io_util
     8    com_err
     9    [user application procedure]
     a    [user data segment]
     b    map_param_Ø1
     c    [menu tree data structure]
     d    [unavailable to user]
     e    linkage [and storage for based]
     f    [storage for automatic variables]
```

The use package terminates each procedure on the menu tree after that particular program has returned. However, it is the user's responsibility to terminate procedures that are called by procedures on the menu-tree.

3.3.3 Communication between programs

It is very common that programs will need to share data, or need a global variable. The menu tree works against this to some extent because it calls separate programs without passing arguments. However, data can be shared in this manner: place all data in a based pll structure; initiate

this segment whose base address is the pointer that addresses the structure. The MagicSix/pll operating environment has a procedure designed expressly for this purpose. It is hcs$initiate_with_options, and allows a segment to be explicity assigned to any one of the core addresses in the above schedule.

3.4 FURTHER WORK

3.4.1 Improved error-handling

The menu program would be a far more useful tool if it flagged syntax errors rather than just blew up when it encountered them. Perhaps a separate program that previewed the text file for proper form before it was handed to the menu program would be the best way to accomplish thsi at this point. At any rate, the menu syntax is simple enough at this rate so that finding mistakes has been a simple matter.

3.4.2 More general menu formats

In this first version, menus build up from the lower left hand corner of the screen. It would be interesting to see if menus would work better if they built from top to bottom as well as bottom to top, and from right to left as well as left to to right. Perhaps ultimately, menus could be freed from being straight lists and arrays: they might simply be linked lists of arbitrarily-placed boxes on the screen.

3.4.3 Dynamic menus

This area still has a lot of interesting work yet to be done. The menu-tree would be far more useful if it were a dynamic structure, which changed as the user used it. It would permit changing the visual format of the menu while it was being used, menus that called other menus, display and generation of data as well as the pre-programmed calling of programs.

4 TYPE DIGITIZATION

4.1 THE IDEAS

4.1.1 Type digitization techniques

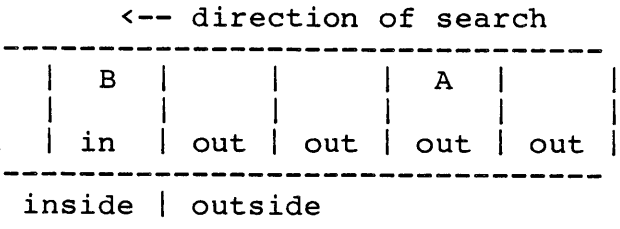
4.1.1.1 Turning a visual letterform into numbers can be done in a number of different ways. Bit maps record the light/dark values at a two-dimensional array of locations in a rectangle which encloses a type form. The light/dark values may be single (Ø or 1) or multiple bit (Ø to 8, or Ø to 255, for example). Bit maps are efficient at small sizes, but quickly grow to unwieldly sizes. Run codes slice through letterforms and record the number of same light/dark values in a sequence. This method produces a more compact code than bit maps, but takes a slightly more complex algorithm to reconstruct the letterform.

4.1.1.2 The most efficient codes in terms of storage are edge-codes, which themselves only with the contour of the letterform. Spline edge codes store the edge as a series of control points. The shape is reconstructed by fitting

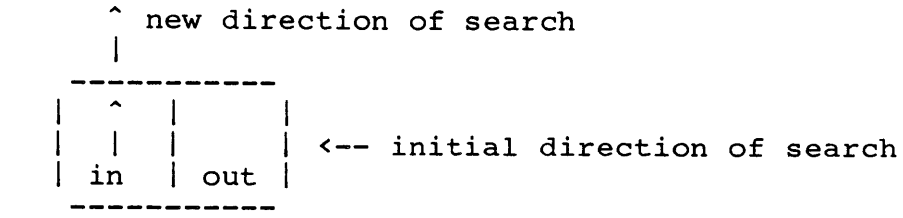
the control points into equations which describe the lines that make contour. Somewhat simpler are chain codes, record unit x and y increments as the shape of the edge is traced. This chain code technique is the method used and explained here.

4.1.2 An edge-detecting routine

4.1.2.1 The edge of the contour is found by starting outside the contour (at A, for example, in the following diagram) and reading the color along a row of pixels. The first pixel lighter than middle grey is the origin of the contour (at B). (Other threshold values might be used, depending on the nature of the image).

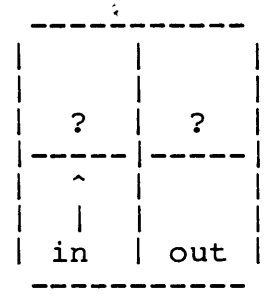


4.1.2.2 Next, the direction of the search is rotated 90 degrees clockwise, with this result: the pixel to the right of the origin is known to be outside the origin because its value was tested as a part of the procedure.

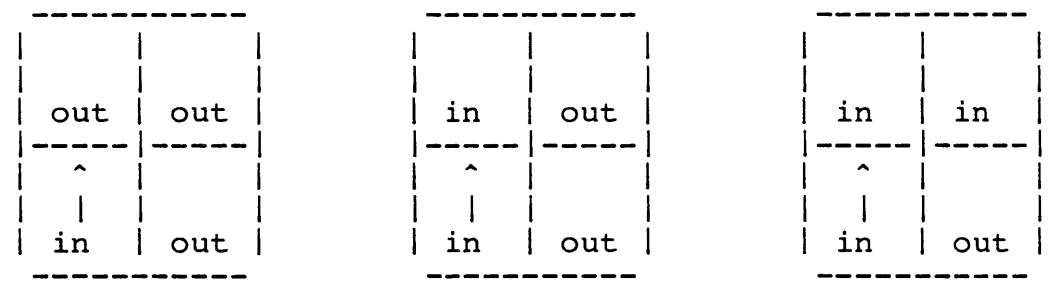


4.1.2.3 The process is now to advance along the contour, always moving into a new cell whose right-hand neighbor is outside the contour. This is a necessary and sufficient condition to ensure that a complete bounded area will be recorded.(It is analogous walking through a maze with one hand always on the wall to ensure that you trace the whole maze and exit where you entered.) As each move is made, a record is kept of x- and y-increments, known as a chain code.

4.1.2.4 The following diagram shows the questions to be answered at each step along the edge: is the pixel directly forward in our out? is the pixel forward and to the left in or out?



4.2.1.5 These are the three possible ways these questions might be answered:



(1) (2) (3)

4.1.2.6 In case (1) we have reached an outside corner. It is necessary to rotate our direction of search 90 degrees counterclockwise. In case (2) the edge is straight, and we advance straight ahead. In case (3) we are on an inside corner. We must advance one unit ahead, one unit to the right, and then turn 90 degrees clockwise. These tests and moves are repeatedly applied until we arrive back at the pixel we started at.

4.2.1.7 The actual code which does the checking described above has the name "check_ahead" and an algorithm which may be written in english as follows:

> If the pixel directly ahead is out then change direction of search 90 counterclockwise, (no need to add a link to the chain code) and start test over again.
>
> Otherwise pixel directly ahead is in (but we don't yet know whether it's on the edge or in the interior). Therefore, make the following test:
>
> > If the pixel located forward and to the right is out, then advance forward, add a link to the chain code, and start test over again.
> >
> > Otherwise the pixel located forward and to the right is in. Advance forward and to right, add a link to the chain code, change direction of search 90 clockwise, and start the test over again.
>
> Do this until the origin is reached again.

4.1.3 Chain coding

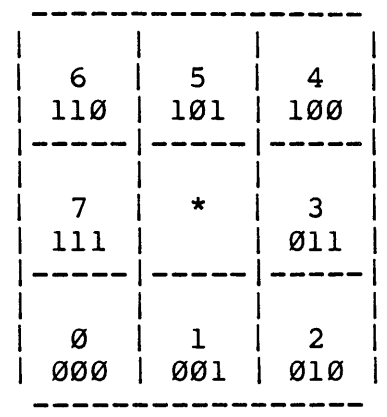
4.1.3.1 The links added to the chain code will be of this nature: Whenever you add a link to the chain code, it will be a four-bit string, two bits to record the x-increments and two bits to record the y-increments, the two-bit codes being:

'01'b	+1 unit
'00'b	0 units
'11'b	-1 unit

4.1.3.2 Two additional codes complete the set: '0000'b means end of a contour (no more x and y increments), '1010'b means end of a set of contours, should the character require more than one contour.

4.1.4 Alternate chain coding routine

4.1.4.1 An even more efficient code consists of three bits per link, and requires that the number of links in a contour be stored separately rather than being flagged in one of the links as in the first example.

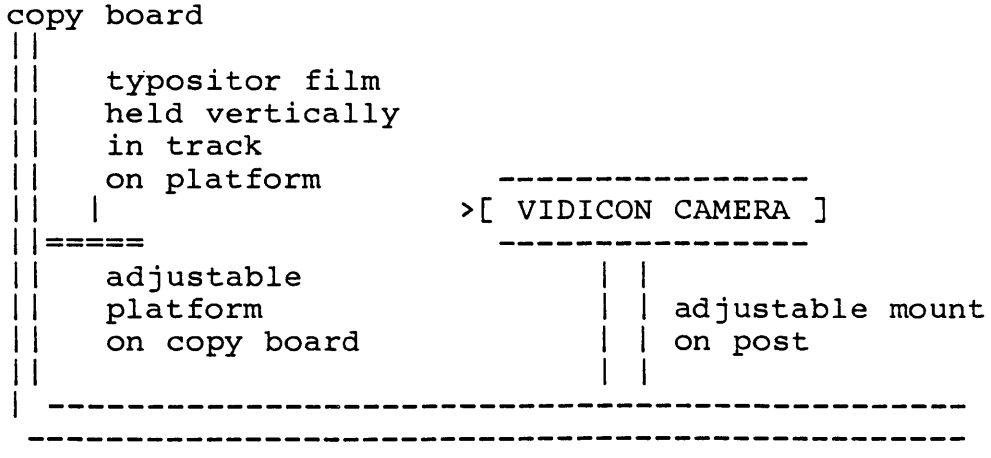


4.1.4.2 In this method, the current point is assumed to be in the center (*) of a 9-unit matrix. The next point on the contour can be in any one of 8 positions, which can be recorded by the specific 3-bit code shown.

4.2 THE CONTOUR PACKAGE: A USER MANUAL

4.2.1 Description the font gun

4.2.1.1 The font gun has four elements. The main structural element is a polaroid copy stand oriented horizontally (the post is horizontal, the copy board vertical). A vidicon digitizing camera is attached to a mount on the post so that its distance from the copy board may be adjusted. A platform is attached to the copy board with two adjustable screws so that its vertical position may be precisely adjusted.



(polaroid copy board and post in horizontal position)

4.2.1.2 The final element is a Visual Graphics Corporation Typositor type font, a filmstrip about 2 inches wide and 1Ø feet in length, which carries negative images of a whole font of characters with very precisely aligned baselines. The quality of the film negative font images fonts on the Visual Graphics Typositor is such that a single character can fill the screen with no appreciable loss of quality.

4.2.1.3 The film is held at right angles to the camera lens in a track on the platform, so there is no keystoning, and the two screw adjustments at either end of the platform make it possible to align horizontal letterstrokes with raster lines of the camera. A white field, about two inches behind the film, illuminated with a 15-watt bulb, provides sufficiently even back-lighting.

4.2.2 Setup to digitize a font

4.2.2.1 Thread the typositor film through the plastic tracks on the platform. Turn on the vidicon camera with AGC (automatic gain control) off. Turn on the 15-watt bulb to backlight the typositor strip.

4.2.2.2 At the computer terminal type:

```
tmrq * <cr>              (clear core memory)
cwd >u>type <cr>         (get in the right directory)
contour <cr>             (run the capture program)
```

4.2.2.3 The program will ask for:

```
name of font file
source of font (VGC typositor font code number)
today's date
```

4.2.2.4 Type in a character string for each answer. Don't use spaces; use underbars (_). Alternatively, surround the string with double quotes, and then you can use spaces. The VGC typositor font code number is found at the beginning and end of each film strip. This information is helpful if we ever need to go back to a recapture chararcters from a font.

4.2.2.5 Next the program will list options:

```
v    vidicon input
c    set cap height via tablet
x    set x height via tablet
b    set baseline via tablet
n    set cap-x-baselines numerically via keyboard
d    display cap-x-baseline values at terminal
q    quit
```

4.2.2.6 This is the setup loop. First press "v" to start continuous vidicon digitizing. The type image will appear on the monitor. Do these things now: Adjust size, focus, and pull enough of the font past the lens to see that the lowest descenders and the highest ascenders fit on the screen. Press <spacebar> to stop digitizing.

4.2.2.7 The program will again list the above options, and this time, press "b", "c", or "x", to mark the baseline, x- height or cap-height. Using the puck, press:

z button (yellow)	to set the line at cursor position
button 1 (white)	to bump line down one pixel
button 3 (green)	to bump line up one pixel
button 2 (blue)	to accept line position.

4.2.2.8 Once the baseline, x-height, or capline is set, press "v" to start vidicon again, and pull some more characters across the screen to check the alignment. Use the screws on each end of the platform if adjustment is needed. When alignment is finished, press "d" to display the values of cap-x-baselines at the terminal. Make a note of them -- these numbers will be needed later.

4.2.2.9 An alternative to setting cap-x-baselines via tablet is to press "n" and enter the values numerically at the keyboard. This allows exact correspondence of values between two files rather than having to make visual judgements at the beginning of each file.

4.2.2.1Ø At the conclusion of this setup activity, press "q" to quit setup, and progress to the actual capture of the font.

4.2.3 Capturing the characters

4.2.3.1 The capture loop will automatically start out in a continuous digitizing mode to allow a single character to be positioned in the center of the screen. When this has been done, press < spacebar> to stop digitizing and press button 1 (white) on the puck to begin acumulating character contours. For each contour, position the cursor to the left of an edge and press the z button (yellow). On the monitor which displays overlay plane 1Ø, the contour will be displayed.

4.2.3.2 To complete the capture of a character, press button 4 (green) to indicate that all contours which define a letter have been accumulated. The program will print out some statistics on the letter (maximum width and height, amount of storage, etc.) and will prompt for the character name to be associated with the letter.

4.2.3.3 To indicate that you are done with a file, press button 2 (blue), and the program will redraw all the outlines it has accumulated.

4.3 FURTHER WORK

4.3.1 Processing the raw captured fonts

Using the methods and programs described in the previous sections, it has been possible to capture about 16 fonts at a time of 2-3 hours per font. This of course is the capture of raw font information. The work that remains is to process this data so characters can be called back to the screen in reduced, multiple-bit form to simulate headline and text at any required size and orientation.

4.3.2 Design and editing of new fonts

Once basic typeforms have been captured, the actual shape of the edge can be changed, smoothed, and recoded, using more compact and refined spline techniques. This methods is general enough so that it can apply to to visual forms beyond familiar typeforms: foreign character sets, ideograms, high contrast images, and logotype forms.

5 CONCLUSION

5.1 THE NATURE OF THESE TOOLS

5.1.1 Objects versus pixels

The software tools described here make it possible for the designer to work with objects rather than with pixels. The distinction is important, because it frees him from the tyranny of the device. No longer is he restricted to directly addressing pixels on the screen. The designer can point at and refer to objects (or models of objects) in the real world, with collections of attributes far richer than just color or grey level. This makes the designer no less of a picture-maker, but a worker with an expanded set of opportunities to plan, assemble, control, and refine, and edit.

5.1.2 Linked lists

Central to all of this increased capacity is the use of dynamic data structures and linked lists. The software

tool builder should acquaint himself with these techniques. [3,4] The basic editing routines of inserting, deleting, copying, sorting, and searching seem to apply to all kinds of work dealing with lists of any type of objects.

5.1.3 Tools to build tools

Building a tool means looking at a problem in a general enough way to expose the process required to solve it. Using a tool, once built, makes future work more productive. This effort has proved the value of modular programming, using software tools already built to help build new ones. This is the spirit of Gerstner's "programmes for solutions."

REFERENCES

REFERENCES

[1] Karl Gerstner, Designing Programmes, trans D. Q. Stephenson, (Teufen, Switzerland: Arthur Niggli Ltd., 1968), p.9.

[2] Newman, William M., Principles of Interactive Computer Graphics, (New York: McGraw-Hill, 1979), p.83.

[3] Wirth, Nicklaus, Algorithms + Data Structures = Programs, (Englewood Cliffs, N.J.: Prentice Hall, 1976).

[4] Knuth, Donald E, The Art of Computer Programming, Second Edition, (Reading Massachusetts: Addison-Wesly, 1973).
Volume 1: Fundamental Algorithms
Volume 3: Searching and Sorting

ACKNOWLEDGMENTS

ACKNOWLEDGEMENTS

I want to thank these people:

My family, Marion, Tom, Jay, Zan, and Lee, whose love and support made these two years at M.I.T. possible.

Muriel Cooper, Ron MacNeil at the Visible Language Workshop, and Ken Sloan, Patrick Purcell at the Architecture Machine Group for guidance, help, and occasional impossible demands that spurred progress.

Lorna Shanks at Xerox, Mike Parker at Bitstream, Ed Schwalenberg at Camex, who added a real-world perspective to this effort.

Anne Russell, Rob Haimes, Lee Silverman, good people whose help and counsel has made this work go smoother.

Fellow graduate students, both here and departed.

Sandy, Debbie, Pam, and Bobby Loring, my family away from home, for encouragement and support.

Richard Durling-Shyderoff for innumerable contributions in the realm of philosophy.

APPENDIX I

Source code for map package

```
------------------------------------------------------------------
1   >u>nardy>map>map.bind, 02:49:10 Thursday April 28, 1983
------------------------------------------------------------------

map
;
load_parameters
setup
;
cursor
cursor_vis
cursor_nbr
;
frame_org
frame_ctr
frame_loc
frame_reset
recalc
;
window
port
tablet
;
x_port
y_port
;
int
fp
raw_xyz
io_util
```

```
------------------------------------------------------------------
1   >u>nardy>map>map_dcl.incl.pl1, 02:57:21 Thursday April 28, 1983
------------------------------------------------------------------

declare

     map$load_parmeters  entry,
     map$setup           entry,
     map$cursor          entry (fix(15), fix(15), fix(15)),
     map$cursor_vis      entry (bit(4)),
     map$cursor_nbr      entry (fix(15)),
     map$frame_org       entry (fix(15), fix(15), fix(15)),
     map$frame_ctr       entry (fix(15), fix(15), fix(15)),
     map$frame_loc       entry (fix(15), fix(15), fix(15)),
     map$frame_reset     entry,
     map$tablet          entry (fix(15), fix(15), fix(15), fix(15)),
     map$window          entry (float(23), float(23), float(23), float(23)),
     map$port            entry (fix(15), fix(15), fix(15), fix(15)),
     map$x_port          entry (float(23)) returns (fix(15)),
     map$y_port          entry (float(23)) returns (fix(15)),
     map$int             entry (fix(15), fix(15), fix(15), fix(15), fix(15)),
     map$fp              entry (fix(15), fix(15), float, float, fix(15));
```

```
-------------------------------------------------------------------
1    >u>nardy>map>p_dcl.incl.pl1, 02:58:13 Thursday April 28, 1983
-------------------------------------------------------------------

declare

     p                        pointer,
     1 q                      based (p),
          2 Uwin,                                /* user window, world coord's */
               3 xorg         float(23),
               3 yorg         float(23),
               3 xext         float(23),
               3 yext         float(23),
          2 Uprt              like Uwin,         /* user screen port, scrn c's */
          2 Utab              like Uwin,         /* user tablet port, tab c's  */
          2 win               like Uwin,         /* scaled part of user window */
          2 tab               like Uwin,         /* scaled part of user tablet */
          2 prt               like Uwin,         /* scaled part of user port   */
          2 buf               like Uwin,         /* scaled part of frame buffer*/
          2 cur               like Uwin,         /* cursor port                */
          2 scr               like Uwin,         /* cursor space 0-->511       */
          2 xoffset           float(23),
          2 yoffset           float(23),
          2 scale             float(23),
          2 cur_nbr           fix(15),           /* 1, 2, 3, 4                 */
          2 cur_vis           bit(16);           /* '000x'b4 where x = '4321'  */
```

```
------------------------------------------------------------------
1   >u>nardy>map>load_parameters.pl1, 02:50:29 Thursday April 28, 1983
------------------------------------------------------------------

load_parameters: procedure;

declare
     scs$get_wdir   entry (char(168)vary),
     hcs$append_seg entry (char(168)vary, char(32)vary, fix(31)),
     hcs$initiate_w_options entry (char(168)vary, char(32)vary,
                    char(32)vary, bit(1), pointer, fix(31)),
     dir_name       char(168)vary,
     param_ptr      pointer,
     syscode        fix(31),
     oops           condition;

/*****************************************************************/

syscode_manager: procedure (string);
     declare
          com_error entry options (variable),
          string    char(64)vary;
     call com_error (syscode, "load_parameters", string);
     signal oops;
     end;

on oops begin;
     goto exit;
     end;

unspec (param_ptr) = '000b0000'b4;
call scs$get_wdir (dir_name);
call hcs$append_seg (dir_name, "map_param_01", syscode);
if syscode ^= -14 then do;
     if syscode < 0 then call syscode_manager ("making 'map_param_01'");
     end;
call hcs$initiate_w_options
     (dir_name, "map_param_01", "map_param_01", '1'b, param_ptr, syscode);
if syscode < 0 then call syscode_manager ("initiating 'map_param_01'");

exit:;
end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>setup.pl1, 02:51:30 Thursday April 28, 1983
------------------------------------------------------------------

setup: procedure;

%include p_dcl;

declare
     $load_parameters    entry,
     $frame_org          entry (fix(15), fix(15), fix(15));

/*****************************************************************/

stuff: procedure (x_origin, y_origin, x_extent, y_extent, a_ptr);

     declare
          x_origin       float(23),
          y_origin       float(23),
          x_extent       float(23),
          y_extent       float(23),
          a_ptr          pointer,
          1 a            based (a_ptr),
               2 xorg    float(23),
               2 yorg    float(23),
               2 xext    float(23),
               2 yext    float(23);

     a_ptr->a.xorg = x_origin;
     a_ptr->a.yorg = y_origin;
     a_ptr->a.xext = x_extent;
     a_ptr->a.yext = y_extent;
     end;

/*****************************************************************/

call $load_parameters;
unspec (p) = '000b0000'b4;

call stuff (  0.0,    0.0,   512.0,   512.0, addr (p->q.Uwin));
call stuff (77.0,  601.0,  2048.0,  1536.0, addr (p->q.Utab));
call stuff (  0.0,    0.0,   512.0,   512.0, addr (p->q.Uprt));
call stuff (  0.0,    0.0,   512.0,   512.0, addr (p->q.scr));
call stuff (  0.0,    0.0,   512.0,   512.0, addr (p->q.buf));

p->q.scale     = 1.0;
p->q.cur_nbr   = 1;
p->q.cur_vis   = '0001'b4;

call $frame_org (0, 0, 1);

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>cursor.pl1, Ø2:55:26 Thursday April 28, 1983
------------------------------------------------------------------

cursor: procedure (nbr, x, y);

%include p_dcl;

declare
     (x, y, nbr)    fix(15),
     grin$vis       entry (bit(16)),
     grin$pos       entry (fix(15), fix(15), fix(15));

call grin$vis (p->q.cur_vis);
call grin$pos (p->q.cur_nbr,
     ((x - p->q.buf.xorg) * p->q.scale) + p->q.xoffset,
     ((y - p->q.buf.yorg) * p->q.scale) + p->q.yoffset);

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>cursor_vis.pl1, 02:55:45 Thursday April 28, 1983
------------------------------------------------------------------

cursor_vis: procedure (string4);

%include p_dcl;

declare
     grin$vis       entry (bit(16)),
     string16       bit(16),
     string4        bit(4);

string16 = '0000'b4;
substr (string16, 13, 4) = substr (string4, 1, 4);
call grin$vis (string16);
p->q.cur_vis = string16;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>map>cursor_nbr.pl1, 02:56:13 Thursday April 28, 1983
-------------------------------------------------------------------

cursor_nbr: procedure (number);

%include p_dcl;

declare number fix(15);

p->q.cur_nbr = number;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>frame_org.pl1, 02:59:27 Thursday April 28, 1983
------------------------------------------------------------------

frame_org: procedure (x_origin, y_origin, scale);

%include p_dcl;

declare
     (x_origin, y_origin, scale)   fix (15),
     (x_center, y_center)          fix (15),
     (xgrin, ygrin, zoom_nbr)      fix (15);

declare
     grin$zoom entry (fix(15), fix(15), fix(15)),
     $recalc        entry;

/*****************************************************************/

unspec(p) = '000b0000'b4;

p->q.buf.xorg = x_origin;
p->q.buf.yorg = y_origin;
p->q.buf.xext = 512 / scale;
p->q.buf.yext = p->q.buf.xext;
p->q.scale = scale;
x_center = p->q.buf.xorg + (p->q.buf.xext / 2.0);
y_center = p->q.buf.yorg + (p->q.buf.yext / 2.0);

xgrin = x_center - 1;
ygrin = y_center - 1 - (32 / scale);
if      scale = 1 then zoom_nbr = 0;
else if scale = 2 then zoom_nbr = 1;
else if scale = 4 then zoom_nbr = 2;
else if scale = 8 then zoom_nbr = 3;
call grin$zoom (xgrin, ygrin, zoom_nbr);

call $recalc;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>frame_ctr.pl1, 03:00:45 Thursday April 28, 1983
------------------------------------------------------------------

frame_ctr: procedure (x_center, y_center, scale);

%include p_dcl;

declare
     (x_center, y_center, scale)   fix(15),
     (xgrin, ygrin, zoom_nbr)      fix(15);

declare
     grin$zoom entry (fix(15), fix(15), fix(15)),
     $recalc        entry;

/*****************************************************************/

unspec(p) = '000b0000'b4;

p->q.buf.xext = 512 / scale;
p->q.buf.yext = p->q.buf.yext;
p->q.buf.xorg = x_center - (p->q.buf.xext / 2);
p->q.buf.yorg = y_center - (p->q.buf.yext / 2);
p->q.scale = scale;

xgrin = x_center - 1;
ygrin = y_center - 1 - (32 / scale);
if      scale = 1 then zoom_nbr = 0;
else if scale = 2 then zoom_nbr = 1;
else if scale = 4 then zoom_nbr = 2;
else if scale = 8 then zoom_nbr = 3;
call grin$zoom (xgrin, ygrin, zoom_nbr);

call $recalc;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>map>frame_loc.pl1, 03:01:48 Thursday April 28, 1983
-------------------------------------------------------------------

frame_loc: procedure (x_locus, y_locus, scale);

%include p_dcl;

declare
     (x_locus, y_locus, scale)     fix(15),
     (x_center, y_center)          fix(15),
     (xgrin, ygrin, zoom_nbr)      fix(15);

declare
     grin$zoom entry (fix(15), fix(15), fix(15)),
     $recalc        entry;

declare
     systemao       condition,
     ioa            entry options (variable);

/*****************************************************************/

on systemao begin;
     end;

unspec(p) = '000b0000'b4;

p->q.buf.xext = 512 / scale;
p->q.buf.yext = p->q.buf.xext;
p->q.buf.xorg = x_locus - (((x_locus - p->q.buf.xorg) * p->q.scale) / scale);
p->q.buf.yorg = y_locus - (((y_locus - p->q.buf.yorg) * p->q.scale) / scale);
p->q.scale = scale;
x_center = p->q.buf.xorg + (p->q.buf.xext / 2.0);
y_center = p->q.buf.yorg + (p->q.buf.yext / 2.0);

xgrin = x_center - 1;
ygrin = y_center - 1 - (32 / scale);
if      scale = 1 then zoom_nbr = 0;
else if scale = 2 then zoom_nbr = 1;
else if scale = 4 then zoom_nbr = 2;
else if scale = 8 then zoom_nbr = 3;
call grin$zoom (xgrin, ygrin, zoom_nbr);

call ioa ("
     buf.xorg = ^f, buf.yorg = ^f, buf.yext = ^f, buf.yorg = ^f",
     p->q.buf.xorg, p->q.buf.yorg, p->q.buf.xext, p->q.buf.yorg);
call ioa ("x_center = ^4i, y_center = ^4i", x_center, y_center);
call ioa ("    xgrin = ^4i,     ygrin = ^4i", xgrin, ygrin);

call $recalc;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>recalc.pl1, 03:15:50 Thursday April 28, 1983
------------------------------------------------------------------

recalc: procedure;

%include p_dcl;

declare
     (buf_lox,  buf_loy,  buf_hix,  buf_hiy)      float(23),
     (Uprt_lox, Uprt_loy, Uprt_hix, Uprt_hiy)     float(23),
     (prt_lox,  prt_loy,  prt_hix,  prt_hiy)      float(23),
     prt_ext                                      float(23),
     integer_part                                 fix(15);

declare
     ioa             entry options (variable),
     systemao        condition;

/*****************************************************************/

on systemao begin;
     end;

unspec (p) = '000b0000'b4;

if p->q.buf.xorg > 0 then do;
     integer_part = p->q.buf.xorg;
     p->q.xoffset = (p->q.buf.xorg - integer_part) * p->q.scale;
     end;
else do;
     integer_part = p->q.buf.xorg;
     p->q.xoffset = (1.0 - (integer_part - p->q.buf.xorg)) * p->q.scale;
     end;

if p->q.buf.yorg > 0 then do;
     integer_part = p->q.buf.yorg;
     p->q.yoffset = (p->q.buf.yorg - integer_part) * p->q.scale;
     end;
else do;
     integer_part = p->q.buf.yorg;
     p->q.yoffset = (1.0 - (integer_part - p->q.buf.yorg)) * p->q.scale;
     end;

/* clamp new prt edges to buf edges, if necessary */
/* first convert org-ext to lo-hi */
/* buf (part displayed) may go outside of physical fb limits     */
/* Uprt never goes outside fb limits, however, so clamping works ok   */

if p->q.buf.xext > 0 then do;
     buf_lox = p->q.buf.xorg;
     buf_hix = p->q.buf.xorg + p->q.buf.xext - 1.0;
     end;
```

```
------------------------------------------------------------------
2   >u>nardy>map>recalc.pl1
------------------------------------------------------------------

else do;
     buf_hix = p->q.buf.xorg;
     buf_lox = p->q.buf.xorg + p->q.buf.xext + 1.0;
     end;

if p->q.buf.yext > 0 then do;
     buf_loy = p->q.buf.yorg;
     buf_hiy = p->q.buf.yorg + p->q.buf.yext - 1.0;
     end;
else do;
     buf_hiy = p->q.buf.yorg;
     buf_loy = p->q.buf.yorg + p->q.buf.yext + 1.0;
     end;

if p->q.Uprt.xext = 0 then do;
     end;
else if p->q.Uprt.xext > 0 then do;
     Uprt_lox = p->q.Uprt.xorg;
     Uprt_hix = p->q.Uprt.xorg + p->q.Uprt.xext - 1.0;
     end;
else do;
     Uprt_hix = p->q.Uprt.xorg;
     Uprt_lox = p->q.Uprt.xorg + p->q.Uprt.xext + 1.0;
     end;

if p->q.Uprt.yext = 0 then do;
     end;
else if p->q.Uprt.yext > 0 then do;
     Uprt_loy = p->q.Uprt.yorg;
     Uprt_hiy = p->q.Uprt.yorg + p->q.Uprt.yext - 1.0;
     end;
else do;
     Uprt_hiy = p->q.Uprt.yorg;
     Uprt_loy = p->q.Uprt.yorg + p->q.Uprt.yext + 1.0;
     end;

prt_ext = 1.0; /* just some value that is not 0.0 */

if buf_hix > Uprt_hix then do;
     if buf_lox > Uprt_hix then prt_ext = 0.0;
     else prt_hix = Uprt_hix;
     end;
else prt_hix = buf_hix;

if buf_hiy > Uprt_hiy then do;
     if buf_loy > Uprt_hiy then prt_ext = 0.0;
     else prt_hiy = Uprt_hiy;
     end;
else prt_hiy = buf_hiy;
```

```
-----------------------------------------------------------------------
3    >u>nardy>map>recalc.pl1
-----------------------------------------------------------------------


if buf_lox < Uprt_lox then do;
     if buf_hix < Uprt_lox then prt_ext = 0.0;
     else prt_lox = Uprt_lox;
     end;
else prt_lox = buf_lox;

if buf_loy < Uprt_loy then do;
     if buf_hiy < Uprt_loy then prt_ext = 0.0;
     else prt_loy = Uprt_loy;
     end;
else prt_loy = buf_loy;

/* convert lo-hi to org-ext and assign new values to structure */

if prt_ext = 0.0 then do;
     p->q.prt.xext = 0.0;
     p->q.prt.yext = 0.0;
     end;
else do;
     if p->q.Uprt.xext < 0 then do;
          p->q.prt.xorg = prt_hix;
          p->q.prt.xext = prt_lox - prt_hix - 1.0;
          end;
     else do;
          p->q.prt.xorg = prt_lox;
          p->q.prt.xext = prt_hix - prt_lox + 1.0;
          end;
     if p->q.Uprt.yext < 0 then do;
          p->q.prt.yorg = prt_hiy;
          p->q.prt.yext = prt_loy - prt_hiy - 1.0;
          end;
     else do;
          p->q.prt.yorg = prt_loy;
          p->q.prt.yext = prt_hiy - prt_loy + 1.0;
          end;
     end;

/* now figure the origins and extents */
/* prt/buf = tab/Utab = cur/scr; prt/Uprt = win/Uwin */
/* the equation is a/b = c/d.  solve for c = (a/b)*d */

p->q.tab.xorg = p->q.Utab.xorg +
     (((p->q.prt.xorg - p->q.buf.xorg) / p->q.buf.xext) * p->q.Utab.xext);
p->q.tab.yorg = p->q.Utab.yorg +
     (((p->q.prt.yorg - p->q.buf.yorg) / p->q.buf.yext) * p->q.Utab.yext);
p->q.tab.xext = (p->q.prt.xext / p->q.buf.xext) * p->q.Utab.xext;
p->q.tab.yext = (p->q.prt.yext / p->q.buf.yext) * p->q.Utab.yext;
p->q.cur.xorg = p->q.scr.xorg +
```

```
------------------------------------------------------------------
4    >u>nardy>map>recalc.pl1
------------------------------------------------------------------

     (((p->q.prt.xorg - p->q.buf.xorg) / p->q.buf.xext) * p->q.scr.xext);
p->q.cur.yorg = p->q.scr.yorg +
     (((p->q.prt.yorg - p->q.buf.yorg) / p->q.buf.yext) * p->q.scr.yext);
p->q.cur.xext = (p->q.prt.xext / p->q.buf.xext) * p->q.scr.xext;
p->q.cur.yext = (p->q.prt.yext / p->q.buf.yext) * p->q.scr.yext;

p->q.win.xorg = p->q.Uwin.xorg +
     (((p->q.prt.xorg - p->q.Uprt.xorg) / p->q.Uprt.xext) * p->q.Uwin.xext);
p->q.win.yorg = p->q.Uwin.yorg +
     (((p->q.prt.yorg - p->q.Uprt.yorg) / p->q.Uprt.yext) * p->q.Uwin.yext);
p->q.win.xext = (p->q.prt.xext / p->q.Uprt.xext) * p->q.Uwin.xext;
p->q.win.yext = (p->q.prt.yext / p->q.Uprt.yext) * p->q.Uwin.yext;

end;

/*
proportions for equations:
(buf' - bufxorg') / bufxext' = (buf - bufxorg) / bufxext
*/
```

```
-------------------------------------------------------------------
1   >u>nardy>map>window.pl1, 03:09:04 Thursday April 28, 1983
-------------------------------------------------------------------

window: procedure (x_origin, y_origin, x_extent, y_extent);

%include p_dcl;

declare
     $recalc   entry;

declare
     x_origin  float(23),
     y_origin  float(23),
     x_extent  float(23),
     y_extent  float(23);

unspec (p) = '000b0000'b4;

p->q.Uwin.xorg = x_origin;
p->q.Uwin.yorg = y_origin;
p->q.Uwin.xext = x_extent;
p->q.Uwin.yext = y_extent;

call $recalc;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>port.pl1, 03:09:54 Thursday April 28, 1983
------------------------------------------------------------------

port: procedure (x_origin, y_origin, x_extent, y_extent);

%include p_dcl;

declare
     $recalc   entry;

declare
     x_origin  fix(15),
     y_origin  fix(15),
     x_extent  fix(15),
     y_extent  fix(15);

unspec (p) = '000b0000'b4;

p->q.Uprt.xorg = x_origin;
p->q.Uprt.yorg = y_origin;
p->q.Uprt.xext = x_extent;
p->q.Uprt.yext = y_extent;

call $recalc;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>map>tablet.pl1, Ø3:13:Ø4 Thursday April 28, 1983
-------------------------------------------------------------------

tablet: procedure (x_origin, y_origin, x_extent, y_extent);

%include p_dcl;

declare
     $recalc   entry;

declare
     x_origin  fix(15),
     y_origin  fix(15),
     x_extent  fix(15),
     y_extent  fix(15);

unspec (p) = '000b0000'b4;

p->q.Utab.xorg = x_origin;
p->q.Utab.yorg = y_origin;
p->q.Utab.xext = x_extent;
p->q.Utab.yext = y_extent;

call $recalc;

end;
```

```
-------------------------------------------------------------------
1    >u>nardy>map>x_port.pl1, 03:13:41 Thursday April 28, 1983
-------------------------------------------------------------------

x_port: procedure (window_x) returns (fix(15));

%include p_dcl;

declare window_x float(23);

unspec (p) = '000b0000'b4;

return (p->q.prt.xorg +
     (((window_x - p->q.win.xorg) / p->q.win.xext) * p->q.prt.xext));

end;
```

```
------------------------------------------------------------------
1    >u>nardy>map>y_port.pl1, 03:14:06 Thursday April 28, 1983
------------------------------------------------------------------

y_port: procedure (window_y) returns (fix(15));

%include p_dcl;

declare window_y float(23);

unspec (p) = '000b0000'b4;

return (p->q.prt.yorg +
     (((window_y - p->q.win.yorg) / p->q.win.yext) * p->q.prt.yext));

end;
```

```
------------------------------------------------------------------
1   >u>nardy>map>int.pl1, 03:21:50 Thursday April 28, 1983
------------------------------------------------------------------

int: procedure (port_x, port_y, int_x, int_y, z);

%include p_dcl;

declare
     grin$vis       entry (bit(16)),
     grin$pos       entry (fix(15), fix(15), fix(15)),
     $raw_xyz       entry (fix(15), fix(15), fix(15));

declare
     (x, y, z)      fix(15),
     port_x         fix(15),
     port_y         fix(15),
     curs_x         fix(15),
     curs_y         fix(15),
     int_x          fix(15),
     int_y          fix(15),
     float_x        float(23),
     float_y        float(23);

/*****************************************************************/

unspec (p) = '000b0000'b4;
call $raw_xyz (x, y, z);

if z => 0 then do;

     float_x = x;
     float_y = y;

     int_x = p->q.win.xorg +
          (((float_x - p->q.tab.xorg) / p->q.tab.xext) * p->q.win.xext);
     int_y = p->q.win.yorg +
          (((float_y - p->q.tab.yorg) / p->q.tab.yext) * p->q.win.yext);

     float_x = int_x;
     float_y = int_y;

     port_x = p->q.prt.xorg +
          (((float_x - p->q.win.xorg) / p->q.win.xext) * p->q.prt.xext);
     port_y = p->q.prt.yorg +
          (((float_y - p->q.win.yorg) / p->q.win.yext) * p->q.prt.yext);

     call grin$vis (p->q.cur_vis);
     call grin$pos (p->q.cur_nbr,
          ((port_x - p->q.buf.xorg) * p->q.scale) + p->q.xoffset,
          ((port_y - p->q.buf.yorg) * p->q.scale) + p->q.yoffset);
     end;
else do;
```

```
------------------------------------------------------------------------
2    >u>nardy>map>int.pll
------------------------------------------------------------------------

     call grin$vis ('0000'b4);
     port_x = 0;
     port_y = 0;
     int_x = 0;
     int_y = 0;
     end;

end;

/*
proportions underlying equations:
(tabx - tabxorg) / tabxext = (winx - winxorg) / winxext
(prtx - prtxorg) / prtxext = (winx - winxorg) / winxext
(curx - curxorg) / curxext = (winx - winxorg) / winxext
*/
```

```
------------------------------------------------------------------
1   >u>nardy>map>fp.pl1, 03:19:42 Thursday April 28, 1983
------------------------------------------------------------------

fp: procedure (port_x, port_y, fp_x, fp_y, z);

%include p_dcl;

declare
     grin$vis       entry (bit(16)),
     grin$pos       entry (fix(15), fix(15), fix(15)),
     $raw_xyz       entry (fix(15), fix(15), fix(15));

declare
     (x, y, z)      fix(15),
     port_x         fix(15),
     port_y         fix(15),
     curs_x         fix(15),
     curs_y         fix(15),
     fp_x           float(23),
     fp_y           float(23),
     float_x        float(23),
     float_y        float(23);

/*****************************************************************/

unspec (p) = '000b0000'b4;
call $raw_xyz (x, y, z);

if z => 0 then do;

     float_x = x;
     float_y = y;

     fp_x = p->q.win.xorg +
          (((float_x - p->q.tab.xorg) / p->q.tab.xext) * p->q.win.xext);
     fp_y = p->q.win.yorg +
          (((float_y - p->q.tab.yorg) / p->q.tab.yext) * p->q.win.yext);

     port_x = p->q.prt.xorg +
          (((float_x - p->q.tab.xorg) / p->q.tab.xext) * p->q.prt.xext);
     port_y = p->q.prt.yorg +
          (((float_y - p->q.tab.yorg) / p->q.tab.yext) * p->q.prt.yext);

     call grin$vis (p->q.cur_vis);
     call grin$pos (p->q.cur_nbr,
          ((port_x - p->q.buf.xorg) * p->q.scale) + p->q.xoffset,
          ((port_y - p->q.buf.yorg) * p->q.scale) + p->q.yoffset);
     end;

else do;
     call grin$vis ('0000'b4);
     port_x = 0;
```

```
------------------------------------------------------------------------
2    >u>nardy>map>fp.pl1
------------------------------------------------------------------------

      port_y = 0;
      fp_x = 0.0;
      fp_y = 0.0;
      end;

end;
```

```
-------------------------------------------------------------------
1    >u>nardy>map>raw_xyz.pl1, 03:23:58 Thursday April 28, 1983
-------------------------------------------------------------------

raw_xyz: procedure (x, y, z);

%include p_dcl;

declare
     $ss             entry (bit(16), bit(8)),
     $wd             entry (bit(16), bit(8)),
     $rd             entry (bit(16), bit(8));

declare

     /*              i/o ports currently invert data from/to the tablet,      */
     /*              so active state of bits is '0'b                          */
     next_byte       bit(8) aligned init ('01101110'b), /* next_byte set      */
     byte_received   bit(8) aligned init ('10101110'b), /* byte_rec'd set     */
     reset_nbbr      bit(8) aligned init ('11101110'b), /* nb&br reset        */
     garbage_byte    bit(8),
     first_byte      bit(8),
     check_byte      bit(8),
     raw_data [1:5] bit(8), /* holds all five bytes before converting to fix */
     tab_addr        bit(16) init ('00aa'b4), /* address of tablet port       */

     i               fix(15),
     x               fix(15),
     y               fix(15),
     z               fix(15),
     delay           fix(15), /* delay loop counter                          */
     duration        fix(15), /* length of delay                             */
     far_field       fix(15), /* cursor is in far field if > 500             */

     xbits           bit(16) defined x,
     ybits           bit(16) defined y,
     zbits           bit(16) defined z;

/*****************************************************************/

unspec (p) = '000b0000'b4;
duration = 45;

call $ss (tab_addr, check_byte);
if check_byte = '04'b4 then do;
     x = 0; y = 0; zbits = 'fe'b4;
     return;
     end;

do check_byte = 'ff'b4 while (substr (check_byte, 1, 1) ^= '0'b);

     do first_byte = 'ff'b4 while (substr (first_byte, 1, 1) ^= '0'b);
          call $wd (tab_addr, next_byte);
```

```
2    >u>nardy>map>raw_xyz.pl1
```

```
          far_field = 0;
          do garbage_byte = 'ff'b4 while (substr (garbage_byte, 2, 1));
               call $rd (tab_addr, garbage_byte);
               far_field = far_field + 1;
               if far_field > 500 then do;
                    x = 0;
                    y = 0;
                    zbits = 'ffff'b4; /* won't take a fixed number. why? */
                    return;
                    end;
               end;
          do delay = 1 to duration;
               end;
          call $wd (tab_addr, reset_nbbr);
          call $rd (tab_addr, first_byte);
          call $wd (tab_addr, byte_received);
          do garbage_byte = '00'b4 while (^substr (garbage_byte, 2, 1));
               call $rd (tab_addr, garbage_byte);
               end;
          call $wd (tab_addr, reset_nbbr);
          end;

     raw_data [1] = first_byte;
     do i = 2 to 5;
          call $wd (tab_addr, next_byte);
          do garbage_byte = 'ff'b4 while (substr (garbage_byte, 2, 1));
               call $rd (tab_addr, garbage_byte);
               end;
          do delay = 1 to duration;
               end;
          call $wd (tab_addr, reset_nbbr);
          call $rd (tab_addr, raw_data [i]);
          call $wd (tab_addr, byte_received);
          do garbage_byte = '00'b4 while (^substr (garbage_byte, 2, 1));
               call $rd (tab_addr, garbage_byte);
               end;
          call $wd (tab_addr, reset_nbbr);
          end;

     call $wd (tab_addr, next_byte);
     far_field = 0;
     do garbage_byte = 'ff'b4 while (substr (garbage_byte, 2, 1));
          call $rd (tab_addr, garbage_byte);
          far_field = far_field + 1;
          if far_field > 500 then do;
               x = 0;
               y = 0;
               zbits = 'ffff'b4;
               return;
```

```
-------------------------------------------------------------------
3   >u>nardy>map>raw_xyz.pl1
-------------------------------------------------------------------

               end;
          end;
     do delay = 1 to duration;
          end;
     call $wd (tab_addr, reset_nbbr);
     call $rd (tab_addr, check_byte);
     call $wd (tab_addr, byte_received);
     do garbage_byte = '00'b4 while (^substr (garbage_byte, 2, 1));
          call $rd (tab_addr, garbage_byte);
          end;
     call $wd (tab_addr, reset_nbbr);

     end;

xbits = 'ffff'b4;
ybits = 'ffff'b4;
zbits = 'ffff'b4;

substr (zbits, 13, 4) = substr (raw_data [1], 3, 4);
substr (xbits, 11, 6) = substr (raw_data [2], 3, 6);
substr (xbits,  5, 6) = substr (raw_data [3], 3, 6);
substr (ybits, 11, 6) = substr (raw_data [4], 3, 6);
substr (ybits,  5, 6) = substr (raw_data [5], 3, 6);

xbits = ~xbits;
ybits = ~ybits;
zbits = ~zbits;

if p->q.tab.xext > 0 then do;
     if x < p->q.tab.xorg then x = p->q.tab.xorg;
     else if x > p->q.tab.xorg + p->q.tab.xext - 1
          then x = p->q.tab.xorg + p->q.tab.xext - 1;
     end;
else do;
     if x > p->q.tab.xorg then x = p->q.tab.xorg;
     else if x < p->q.tab.xorg + p->q.tab.xext - 1
          then x = p->q.tab.xorg + p->q.tab.xext - 1;
     end;

if p->q.tab.yext > 0 then do;
     if y < p->q.tab.yorg then y = p->q.tab.yorg;
     else if y > p->q.tab.yorg + p->q.tab.yext - 1
          then y = p->q.tab.yorg + p->q.tab.yext - 1;
     end;
else do;
     if y > p->q.tab.yorg then y = p->q.tab.yorg;
     else if y < p->q.tab.yorg + p->q.tab.yext - 1
          then y = p->q.tab.yorg + p->q.tab.yext - 1;
     end;
```

```
-----------------------------------------------------------------
4    >u>nardy>map>raw_xyz.pll
-----------------------------------------------------------------

end;
```

APPENDIX II

Source code for menu package

```
------------------------------------------------------------------
1   >u>nardy>menu>menu.bind, 14:36:47 Tuesday April 26, 1983
------------------------------------------------------------------

menu
;
;                       (menu: entry) calls all other programs:
menu_creator
;
;                       file handling procedures:
create_menu_seg
init_text_seg
;
;                       chops up menu into separate sections:
part_finder
;
;                       make linked lists and
;                       fill in menu data structure:
prompt_list_maker
port_maker
tree_maker
add_tree_node
option_maker
figure_xy
;
;                       read data structure to confirm
;                       data is saved properly
prompt_list_writer
port_writer
tree_writer
```

```
------------------------------------------------------------------
1   >u>nardy>menu>menu_dcl.incl.pl1, 14:41:51 Tuesday April 26, 1983
------------------------------------------------------------------

declare

     menu_ptr             pointer,
     1 menu_seg           based (menu_ptr),
          2 leaf_prefix   bit(1),
          2 pop_up_mode   bit(1),
          2 menu_color,
               3 ov8      fix(15),
               3 ov9      fix(15),
               3 ov10     fix(15),
          2 ground_port,
               3 x_abs    fix(15),
               3 y_abs    fix(15),
               3 x_rel    fix(15),
               3 y_rel    fix(15),
               3 x_ext    fix(15),
               3 y_ext    fix(15),
               3 x_ch     fix(15),
               3 y_ch     fix(15),
               3 w_ch     fix(15),
               3 h_ch     fix(15),
               3 outline  fix(15),
               3 txtmode  bit(16),
               3 planes   bit(16),
          2 menu_port     like ground_port,
          2 item_port     like ground_port,
          2 title_port    like ground_port,
          2 prompt_port   like ground_port,
          2 root_data     pointer,
          2 root_menu     pointer,
          2 menu_area     area (16000);
```

```
-------------------------------------------------------------------
1   >u>nardy>menu>menu_item_dcl.incl.pl1, 14:42:44 Tuesday April 26, 1983
-------------------------------------------------------------------

declare

     new_menu            pointer,
     1 m                 based (new_menu),
          2 first_item   pointer,
          2 live_item    pointer,

     to                  pointer,
     1 item              based (to),
          2 label        pointer,
          2 content      pointer,
          2 right        pointer,
          2 left         pointer,
          2 x            fix(15),
          2 y            fix(15);
```

```
------------------------------------------------------------------
1   >u>nardy>menu>menu_creator.pl1, 14:37:52 Tuesday April 26, 1983
------------------------------------------------------------------

menu_creator: procedure;

menu: entry ();

%include menu_dcl;
%include menu_item_dcl;

declare
     (answer)             char(1)vary,
     (ioa, askn)          entry options (variable),

     tmr                  entry options (variable),
     scs$get_arg_count    entry (fix(15)),
     scs$get_arg_info     entry (fix(15), bit(16), fix(15), pointer),
     scs$expand_path      entry (char(168)vary, char(168)vary, char(32)vary,
                               fix(31)),
     hcs$terminate        entry (pointer),

     $init_text_seg       entry (char(168)vary, char(32)vary, pointer, fix(15)),
     $create_menu_seg     entry (char(168)vary, char(32)vary, pointer),
     $part_finder         entry (ptr, ptr, ptr, ptr, fix(15), fix(15), bit(1)),

     $prompt_list_maker   entry (pointer, pointer, pointer, fix(15), fix(15)),
     $port_maker          entry (pointer, pointer, fix(15), fix(15)),
     $trec_maker          entry (pointer, pointer, pointer, fix(15), fix(15)),
     $option_maker        entry (pointer, pointer, fix(15), fix(15)),
     $port_writer         entry (pointer),
     $prompt_list_writer  entry (pointer),
     $tree_writer         entry (pointer),
     $figure_xy           entry (fix(15), fix(15), pointer, pointer),

     parts                fix(15) init (4),
     start                pointer,
     s [1:5]              fixed(15) based (start),
     finish               pointer,
     f [1:5]              fixed(15) based (finish),
     key                  pointer,
     k [1:5]              char(16)vary based (key),

     num_args             fix(15),
     arg_type             bit(16),
     arg_length           fix(15),
     arg_ptr              pointer,
     arg_string           char(32) based (arg_ptr),

     dir_name             char(168)vary,
     entry_name           char(32)vary,
     menu_name            char(32)vary,
     garbage_name         char(32)vary,
```

```
------------------------------------------------------------------
2   >u>nardy>menu>menu_creator.pl1
------------------------------------------------------------------


     text_ptr             pointer,
     head_ptr             pointer,

     i                    fix(15),
     char_count           fix(15),
     syscode              fix(31),

     text_menu_error      bit(1),

     oops                 condition;

/*****************************************************************/

on oops begin;
     goto exit;
     end;

call scs$get_arg_count (num_args);
if num_args = 0 then call askn ("name of menu: ", menu_name);
else do;
     call scs$get_arg_info (1, arg_type, arg_length, arg_ptr);
     menu_name = substr (arg_ptr->arg_string, 1, arg_length);
     end;
if substr (menu_name, length (menu_name) - 4, 5) = ".menu"
     then menu_name = substr (menu_name, 1, length (menu_name) - 5);
entry_name = menu_name || ".menu";
if num_args = 2 then do;
     call scs$get_arg_info (2, arg_type, arg_length, arg_ptr);
     if substr (arg_ptr->arg_string, 1, arg_length) = "special"
          then unspec (menu_ptr) = '000a0000'b4;
     end;
else unspec (menu_ptr) = '000c0000'b4;

call scs$expand_path      (menu_name, dir_name, garbage_name, syscode);
call $init_text_seg       (dir_name, entry_name, text_ptr, char_count);
call $create_menu_seg     (dir_name, menu_name, menu_ptr);

allocate s;
allocate f;
allocate k;

do i = 1 to parts;
     start->s[i]  = -1;
     finish->f[i] = -1;
     end;

key->k[1] = "**PROMPTS";
key->k[2] = "**MENU";
```

```
------------------------------------------------------------------
3    >u>nardy>menu>menu_creator.pl1
------------------------------------------------------------------

key->k[3] = "**OPTIONS";
key->k[4] = "**PORTS";

call $part_finder
     (text_ptr, start, finish, key, parts, char_count, text_menu_error);

if (~ text_menu_error) then do;
     call $prompt_list_maker
          (menu_ptr, text_ptr, head_ptr, start->s[1], finish->f[1]);
     call $prompt_list_writer (head_ptr);
     call $port_maker (menu_ptr, text_ptr, start->s[4], finish->f[4]);
     call $port_writer (menu_ptr);
     call $tree_maker
          (menu_ptr, text_ptr, head_ptr, start->s[2], finish->f[2]);
     call $option_maker (menu_ptr, text_ptr, start->s[3], finish->f[3]);
     call $figure_xy
          (0, -1, menu_ptr->menu_seg.root_menu->m.first_item, menu_ptr);
     call ioa ("^r^r********************* MENU **********************^r");
     call $tree_writer (menu_ptr->menu_seg.root_menu->m.first_item);
     end;

call hcs$terminate (text_ptr);
call hcs$terminate (menu_ptr);

free s;
free f;
free k;

exit:;
end;
```

```
------------------------------------------------------------------
1   >u>nardy>menu>create_menu_seg.pl1, 14:44:17 Tuesday April 26, 1983
------------------------------------------------------------------

create_menu_seg: procedure (dir_name, menu_file, menu_ptr);

declare
     ioa             entry options (variable),
     com_error       entry options (variable),
     hcs$append_seg  entry (char(168)vary, char(32)vary, fix(31)),
     hcs$initiate_w_options
                     entry (char(168)vary, char(32)vary, char(32)vary,
                     bit(1), pointer, fix(31)),
     dir_name        char(168)vary,
     menu_file       char(32)vary,
     menu_ptr        pointer,
     syscode         fix(31),
     oops            condition;

syscode_manager: procedure (string);
     declare string char(64)vary;
     call com_error (syscode, "create_menu_seg", string);
     signal oops;
     end;

on oops begin;
     goto exit;
     end;

call hcs$append_seg (dir_name, menu_file, syscode);
if syscode ^= -14 then do;
     if syscode < 0 then call syscode_manager ("making " || menu_file);
     else call ioa ("--> new menu file '^a' added to directory '^a'",
          menu_file, dir_name);
     end;
else call ioa ("--> menu file '^a' already exists. you are bashing it",
     menu_file);

call hcs$initiate_w_options
     (dir_name, menu_file, menu_file, '1'b, menu_ptr, syscode);
if syscode < 0 then call syscode_manager ("initiating " || menu_file);
else call ioa ("--> menu file '^a' has been initiated at address ^p",
     menu_file, menu_ptr);

exit:;
end;
```

```
------------------------------------------------------------------
1   >u>nardy>menu>init_text_seg.pl1, 14:45:46 Tuesday April 26, 1983
------------------------------------------------------------------

init_text_seg: procedure (dir_name, text_file, text_ptr, char_count);

declare
     ioa             entry options (variable),
     com_error       entry options (variable),
     hcs$get_bit_count entry (pointer, fix(31)),
     hcs$initiate    entry (char(168)vary, char(32)vary, ptr, fix(31)),
     dir_name        char(168)vary,
     text_file       char( 32)vary,
     text_ptr        pointer,
     syscode         fix(31),
     bit_count       fix(31),
     char_count      fix(15),
     oops            condition;

syscode_manager: procedure (string);
     declare string char(64)vary;
     call com_error (syscode, "init_text_seg", string);
     signal oops;
     end;

on oops begin;
     goto exit;
     end;

call hcs$initiate (dir_name, text_file, text_ptr, syscode);
if syscode < 0 then call syscode_manager ("initiating " || text_file);
else call ioa ("--> text file '^a' has been initiated", text_file);
call hcs$get_bit_count (text_ptr, bit_count);
char_count = bit_count / 8;

exit:;
end;
```

```
-------------------------------------------------------------------
1    >u>nardy>menu>part_finder.pl1, 14:46:44 Tuesday April 26, 1983
-------------------------------------------------------------------

part_finder: procedure (text, start, finish, key, parts, char_count, error);

declare

     (ioa, ioan)          entry options (variable),
     (ask, askn)          entry options (variable),

     start                pointer,
     s [1:1]              fix(15) based (start),
     finish               pointer,
     f [1:1]              fix(15) based (finish),
     key                  pointer,
     k [1:1]              char(16)vary based (key),

     text                 pointer,
     ascii_string         char(30000) based (text),
     end_ptr              pointer,
     prev_end_ptr         pointer,

     fix15_ptr            pointer,
     fix15                fix(15) based (fix15_ptr),
     parts                fix(15),
     char_count           fix(15),
     garbage_var          fix(15),
     (i, j)               fix(15),

     comment              bit(1) init ('0'b),
     yes                  bit(1) init ('1'b),
     no                   bit(1) init ('0'b),
     error                bit(1),

     answer               char(1);

/*****************************************************************/

prev_end_ptr = addr(garbage_var);
do i = 1 to char_count;
     if substr (text->ascii_string, i, 1) = "|" then comment = yes;
     else if substr (text->ascii_string, i, 1) = "
" then comment = no;
     if (~ comment) then if substr (text->ascii_string, i, 2) = "**" then do;
          do j = 1 to parts;
               if (substr (text->ascii_string, i, length (key->k [j]))
                    = key->k [j]) then do;
                    start->s [j] = i + length (key->k [j]);
                    end_ptr = addr (finish->f [j]);
                    end;
               end;
          prev_end_ptr->fix15 = i - 1;
```

```
------------------------------------------------------------------
2   >u>nardy>menu>part_finder.pl1
------------------------------------------------------------------

          prev_end_ptr = end_ptr;
          end;
     end;
end_ptr->fix15 = char_count;

error = '0'b;
do i = 1 to parts;
     if start->s[i] > 0 then do;
          call ioa ("start ^i  finish ^i", start->s[i], finish->f[i]);
          call ioan ("^a", key->k[i]);
          call ioan ("^a", substr (text->ascii_string,
               start->s[i], finish->f[i] - start->s[i] + 1));
          call askn ("--> is this part identified ok? (y/n) ", answer);
          if answer = "n" then error = '1'b;
          end;
     end;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>menu>prompt_list_maker.pl1, 14:48:54 Tuesday April 26, 1983
------------------------------------------------------------------

prompt_list_maker: procedure (menu_ptr, text_ptr, head_ptr, start, finish);

%include menu_dcl;

declare

     scs$allocn          entry (fix(31), pointer, area),

     head_ptr            pointer,
     match_ptr           pointer,
     1 match             based (match_ptr),
          2 string       char(32)vary,
          2 data         pointer,
          2 next         pointer,

     text_ptr            pointer,
     ascii_string        char(1) based,
     ascii_array [1:1]   char(1) based,

     start               fix(15),
     finish              fix(15),
     string_start        fix(15),
     string_length       fix(15),
     (i, j)              fix(15),
     fix15               fix(15) based,

     alloc_bytes         fix(31),       /* string_length + 2 */

     one_char            char(1),
     something_there     bit(1),
     yes                 bit(1) init ('1'b),
     no                  bit(1) init ('0'b);

/*****************************************************************/

menu_ptr->menu_seg.menu_area = empty();
head_ptr = null();
something_there = no;
string_length = 0;

do i = start to finish;
     one_char = text_ptr->ascii_array [i];
     if one_char = "[" then do;
          if something_there then do;
               string_length = string_length - 1; /* delete last <cr> */
               alloc_bytes = string_length + 2;
               call scs$allocn (alloc_bytes, match_ptr->match.data,
                    menu_ptr->menu_seg.menu_area);
               match_ptr->match.data->fix15 = string_length;
```

```
------------------------------------------------------------------
2    >u>nardy>menu>prompt_list_maker.pl1
------------------------------------------------------------------

               do j = 1 to string_length;
                    match_ptr->match.data->ascii_array [j + 2] =
                         text_ptr->ascii_array [string_start + j - 1];
                    end;
               end;
          string_start  = i + 1;
          string_length = -1;
          end;
     else if one_char = "]" then do;
          something_there = yes;
          allocate match;
          match_ptr->match.string =
               substr (text_ptr->ascii_string, string_start, string_length);
          match_ptr->match.next = head_ptr;
          head_ptr = match_ptr;
          string_start  = i + 2;
          string_length = -2;
          end;
     string_length = string_length + 1;
     end;

string_length = string_length - 1; /* delete last <cr> */
alloc_bytes = string_length + 2;
call scs$allocn (alloc_bytes, match_ptr->match.data,
     menu_ptr->menu_seg.menu_area);
match_ptr->match.data->fix15 = string_length;
do j = 1 to string_length;
     match_ptr->match.data->ascii_array [j + 2] =
     text_ptr->ascii_array [string_start + j - 1];
     end;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>menu>port_maker.pl1, 14:50:43 Tuesday April 26, 1983
-------------------------------------------------------------------

port_maker: procedure (menu_ptr, text_ptr, start, finish);

%include menu_dcl;

declare

     text_ptr            pointer,
     ascii_array [1:1]   char(1) based (text_ptr),

     start               fix(15),
     finish              fix(15),
     start_line          fix(15),
     (i, j)              fix(15),

     ch                  char(1),
     match_string        char(16)vary,

     done                bit(1),
     yes                 bit(1) init ('1'b),
     no                  bit(1) init ('0'b);

/*****************************************************************/

next_char: procedure;

     i = i + 1;
     ch = text_ptr->ascii_array [i];
     end;

/*****************************************************************/

string_to_fix: procedure returns (fix(15));
/* note: tried pl1 built-in, but it didn't work very well */

     declare fixvar fix(15);

     fixvar = 0;
     do while ((ch ^= " ") & (ch ^= "    "));
          if      ch = "0" then fixvar = 10 * fixvar;
          else if ch = "1" then fixvar = (10 * fixvar) + 1;
          else if ch = "2" then fixvar = (10 * fixvar) + 2;
          else if ch = "3" then fixvar = (10 * fixvar) + 3;
          else if ch = "4" then fixvar = (10 * fixvar) + 4;
          else if ch = "5" then fixvar = (10 * fixvar) + 5;
          else if ch = "6" then fixvar = (10 * fixvar) + 6;
          else if ch = "7" then fixvar = (10 * fixvar) + 7;
          else if ch = "8" then fixvar = (10 * fixvar) + 8;
          else if ch = "9" then fixvar = (10 * fixvar) + 9;
          call next_char;
```

```
------------------------------------------------------------------
2   >u>nardy>menu>port_maker.pll
------------------------------------------------------------------

          end;
     do while ((ch = " ") | (ch = "      "));
          call next_char;
          end;
     return (fixvar);
     end;

/*****************************************************************/

string_to_bit: procedure returns (bit(8));
/* tried pll built-in, but it didn't work very well */
     declare
          k              fix(15),
          bitvar         bit(16);

     bitvar = '0000'b4;
     do k = 8 to 6 by -1;
          if ch = "1" then substr (bitvar, k, 1) = '1'b;
          call next_char;
          end;
     return (bitvar);
     end;

/*****************************************************************/

assign_ten: procedure (port_ptr);

     declare
          port_ptr       pointer,
          1 port         based,
               2 x_abs   fix(15),
               2 y_abs   fix(15),
               2 x_rel   fix(15),
               2 y_rel   fix(15),
               2 x_ext   fix(15),
               2 y_ext   fix(15),
               2 x_ch    fix(15),
               2 y_ch    fix(15),
               2 w_ch    fix(15),
               2 h_ch    fix(15),
               2 outline fix(15),
               2 txtmode bit(16),
               2 planes  bit(16);

     port_ptr->port.x_rel     = string_to_fix ();
     port_ptr->port.y_rel     = string_to_fix ();
     port_ptr->port.x_ext     = string_to_fix ();
     port_ptr->port.y_ext     = string_to_fix ();
     port_ptr->port.x_ch      = string_to_fix ();
```

```
-----------------------------------------------------------------
3   >u>nardy>menu>port_maker.pll
-----------------------------------------------------------------

     port_ptr->port.y_ch      = string_to_fix ();
     port_ptr->port.w_ch      = string_to_fix ();
     port_ptr->port.h_ch      = string_to_fix ();
     port_ptr->port.outline   = string_to_fix ();
     port_ptr->port.planes    = string_to_bit ();
     end;

/*****************************************************************/

start_line = start;
done = no;
do while (done = no);
     j = start_line;
     ch = text_ptr->ascii_array [j];
     do while ((j ~= finish) & (unspec (ch) ~= 13));
          j = j + 1;
          ch = text_ptr->ascii_array [j];
          end;
     if j = finish then done = yes;

     /* find the first non-space/tab char */
     i = start_line;
     ch = text_ptr->ascii_array [i];
     do while ((ch = " ") | (ch = "     "));
          call next_char;
          end;

     if ch ~= "|" then                        /* a comment line     */
          if ch ~= "*" then                   /* begin next section */
               if unspec (ch) ~= 13 then      /* <cr> end of line   */
     do;
          match_string = "";
          do while ((ch ~= " ") & (ch ~="     "));
               match_string = match_string || ch;
               call next_char;
               end;
          do while ((ch = " ") | (ch = "     "));
               call next_char;
               end;

          /* assign structure values depending on match_string */
          if (substr (match_string, 1, 6)) = "ground"
               then call assign_ten (addr(menu_ptr->menu_seg.ground_port));
          else if (substr (match_string, 1, 4)) = "menu"
               then call assign_ten (addr(menu_ptr->menu_seg.menu_port));
          else if (substr (match_string, 1, 4)) = "item"
               then call assign_ten (addr(menu_ptr->menu_seg.item_port));
          else if (substr (match_string, 1, 5)) = "title"
               then call assign_ten (addr(menu_ptr->menu_seg.title_port));
```

```
------------------------------------------------------------------
4    >u>nardy>menu>port_maker.pl1
------------------------------------------------------------------

          else if (substr (match_string, 1, 6)) = "prompt"
               then call assign_ten (addr(menu_ptr->menu_seg.prompt_port));

          end;
     start_line = j + 1;
     end;

/* calculate absolute screen positions for each port */
/* (not necessary to calculate abs for items)        */

menu_ptr->menu_seg.ground_port.x_abs = menu_ptr->menu_seg.ground_port.x_rel;
menu_ptr->menu_seg.ground_port.y_abs = menu_ptr->menu_seg.ground_port.y_rel;

menu_ptr->menu_seg.item_port.x_abs =
     menu_ptr->menu_seg.ground_port.x_abs + menu_ptr->menu_seg.menu_port.x_rel;
menu_ptr->menu_seg.item_port.y_abs =
     menu_ptr->menu_seg.ground_port.y_abs + menu_ptr->menu_seg.menu_port.y_rel;

menu_ptr->menu_seg.menu_port.x_abs =
     menu_ptr->menu_seg.ground_port.x_abs +
          menu_ptr->menu_seg.menu_port.x_rel;
menu_ptr->menu_seg.menu_port.y_abs =
     menu_ptr->menu_seg.ground_port.y_abs +
          menu_ptr->menu_seg.menu_port.y_rel;

menu_ptr->menu_seg.title_port.x_abs =
     menu_ptr->menu_seg.ground_port.x_abs +
          menu_ptr->menu_seg.title_port.x_rel;
menu_ptr->menu_seg.title_port.y_abs =
     menu_ptr->menu_seg.ground_port.y_abs +
          menu_ptr->menu_seg.title_port.y_rel;

menu_ptr->menu_seg.prompt_port.x_abs =
     menu_ptr->menu_seg.ground_port.x_abs +
          menu_ptr->menu_seg.prompt_port.x_rel;
menu_ptr->menu_seg.prompt_port.y_abs =
     menu_ptr->menu_seg.ground_port.y_abs +
          menu_ptr->menu_seg.prompt_port.y_rel;

/* calculate txtmode for ports                          */
/* (txtmode is not used for ground_port or menu_port) */
menu_ptr->ground_port.txtmode = '1111'b4;
menu_ptr->menu_port.txtmode   = '1111'b4;

menu_ptr->item_port.txtmode = '0000'b4;
if menu_ptr->item_port.w_ch > 15 then
     substr (menu_ptr->item_port.txtmode, 6, 1) = '1'b;
if menu_ptr->item_port.h_ch > 23 then
     substr (menu_ptr->item_port.txtmode, 5, 1) = '1'b;
```

```
------------------------------------------------------------------
1   >u>nardy>menu>tree_maker.pl1, 15:52:57 Tuesday April 26, 1983
------------------------------------------------------------------

tree_maker: procedure (menu_ptr, text_ptr, head_ptr, start, finish);

%include menu_dcl;
%include menu_item_dcl;

declare

     (ioa, ioan)          entry options (variable),
     scs$allocn           entry (fix(31), pointer, area),
     $add_tree_node       entry (pointer, pointer, pointer),

     from                 pointer,
     auxiliary            pointer,

     space                char(1) init (" "),
     tab                  char(1) init ("     "),
     ret                  char(1),
     ch                   char(1),

     (i, j)               fix(15),
     menu_level           fix(15),
     space_count          fix(15),
     first_legal          fix(15),
     last_legal           fix(15),
     start                fix(15),
     finish               fix(15),

     (done, quit)         bit(1),
     leaf_bit             bit(1),
     yes                  bit(1) init ('1'b),
     no                   bit(1) init ('0'b),

     text_ptr             pointer,
     ascii_string         char(1) based,
     ascii_array [1:1]    char(1) based,

     label_ptr            pointer,
     content_ptr          pointer,
     head_ptr             pointer;

/*****************************************************************/

next_char: procedure;

     i = i + 1;
     ch = text_ptr->ascii_array [i];
     end;

/*****************************************************************/
```

```
------------------------------------------------------------------
2   >u>nardy>menu>tree_maker.pl1
------------------------------------------------------------------


save_string: procedure (data_ptr);

     declare
          scs$allocn      entry (fix(31), pointer, area),
          data_ptr        pointer,
          alloc_bytes     fix(31),
          fix_num         fix(15),
          fix15           fix(15) based;

     fix_num = last_legal - first_legal + 1;
     alloc_bytes = fix_num + 2;
     call scs$allocn (alloc_bytes, data_ptr, menu_ptr->menu_seg.menu_area);
     data_ptr->fix15 = fix_num;
     do j = first_legal to last_legal;
          if text_ptr->ascii_array [j] = "\\" then
               data_ptr->ascii_array [j - first_legal + 3] = ret;
          else data_ptr->ascii_array [j - first_legal + 3] =
               text_ptr->ascii_array [j];
          end;
     end;

/*****************************************************************/

match_and_save: procedure (data_ptr);

     declare
          data_ptr             pointer,
          (fix_num, j)         fix(15),
          fix15                fix(15) based,
          str_len              fix(15),
          match_ptr            pointer,
          1 match              based (match_ptr),
               2 string        char(32)vary,
               2 data          pointer,
               2 next          pointer,
          match_string         char(32)vary;

     /* you can do this quod allocated vars stick around in      */
     /* the linkage segment (e) after the procedure goes away    */
     str_len = last_legal - first_legal - 1;
     match_string = substr (text_ptr->ascii_string,
          first_legal+1, str_len);
     do match_ptr = head_ptr repeat match_ptr->match.next
          while (substr (match_ptr->match.string, 1, str_len) ^=
               substr (match_string, 1, str_len));
          end;
     data_ptr = match_ptr->match.data;
     end;
```

```
-------------------------------------------------------------------
3   >u>nardy>menu>tree_maker.pll
-------------------------------------------------------------------


/*****************************************************************/

ret = "
";
menu_ptr->menu_seg.root_menu = null();
i = start;
call next_char;
do done = no while (done = no);

     /* determine the menu level by counting spaces/tabs */
     menu_level  = 0;
     space_count = 0;
     do while ((ch = space) | (ch = tab));
          if ch = tab then do;
               menu_level = menu_level + 1;
               space_count = 0;
               end;
          else if ch = space then do;
               space_count = space_count + 1;
               if space_count = 5 then do;
                    menu_level = menu_level + 1;
                    space_count = 0;
                    end;
               end;
          call next_char;
          end;

     /* arrive here knowing menu_level and first non-space/tab char */
     /* if ch is not carriage return or comment marker, work on the line */
     if ch ~= ret then if ch ~= "|" then do;

          /* find last_legal ch of label (delete trailing space/tabs) */
          first_legal = i;
          last_legal  = i;
          leaf_bit = no;
          do quit = no while (quit = no);
               call next_char;
               if      ch = "'" then quit = yes;
               else if ch = '"' then quit = yes;
               else if ch = "[" then quit = yes;
               else if ch = ":" then do;
                    quit = yes;
                    leaf_bit = yes;
                    end;
               else if ch = "|" then do;
                    quit = yes;
                    content_ptr = null();
                    end;
```

```
-----------------------------------------------------------------
4   >u>nardy>menu>tree_maker.pl1
-----------------------------------------------------------------

               else if ch = ret then do;
                    quit = yes;
                    content_ptr = null();
                    end;
               else if ch ~= space then if ch ~= tab then last_legal = i;
               end;

          /* arrive here knowing first_legal ch, last_legal ch   */
          if text_ptr->ascii_array [first_legal] = "["
               then call match_and_save (label_ptr);
          else call save_string (label_ptr);

          /* arrive here with next non-space/tab ch & 3 options: */
          /* 1. content_ptr = null(): don't work on rest of line */
          /* 2. ch = ":": what follows is program name string    */
          /* 3. otherwise ch = "'" or '"' or "[": prompt string  */

          if content_ptr ~= null() then do;
               if ch = ":" then do;
                    /* delete leading space/tabs */
                    call next_char;
                    do while ((ch = space) | (ch = tab));
                         call next_char;
                         end;
                    first_legal = i;
                    do quit = no while (quit = no);
                         call next_char;
                         if      ch = ret    then quit = yes;
                         else if ch = "|"    then quit = yes;
                         else if ch = space  then quit = yes;
                         else if ch = tab    then quit = yes;
                         end;
                    last_legal = i - 1;
                    call save_string (content_ptr);
                    end;
               else do;
                    /* get and save prompt string:
                    you already have taken care of leading space/tabs
                    and you can include trailing ones, but your first char
                    is " or ' or [ and you don't want to include it */
                    first_legal = i;
                    do quit = no while (quit = no);
                         call next_char;
                         if      ch = ret then quit = yes;
                         else if ch = "|" then quit = yes;
                         else if ch = "'" then quit = yes;
                         else if ch = '"' then quit = yes;
                         else if ch = "]" then quit = yes;
                         end;
```

```
------------------------------------------------------------------
5    >u>nardy>menu>tree_maker.pl1
------------------------------------------------------------------

                    last_legal = i;
                    if ch = "]" then call match_and_save (content_ptr);
                    else do;
                         first_legal = first_legal + 1;
                         last_legal = i - 1;
                         call save_string (content_ptr);
                         end;
                    end;
               end;

          /* arrive here with menu_level, label_ptr, content_ptr, leaf_bit */
          call scs$allocn (20, to, menu_ptr->menu_seg.menu_area);
          to->item.y       = menu_level - 1; /* level -1 is not displayed */
          to->item.label   = label_ptr;
          to->item.content = content_ptr;
          if leaf_bit = yes then to->item.right = null();

          if (menu_ptr->menu_seg.root_menu = null()) then do;
               do;
                    call scs$allocn
                    (8, new_menu, menu_ptr->menu_seg.menu_area);
                    new_menu->m.first_item = to;
                    new_menu->m.live_item  = null();
                    end;
               menu_ptr->menu_seg.root_menu = new_menu;
               to->item.x = 0;
               to->item.left = null();
               end;
          else call $add_tree_node (to, from, menu_ptr);
          from = to;
          end;

     /* if necessary mop up rest of chars on line, find car_return */
     do while (ch ~= ret);
          call next_char;
          end;
     if i => finish then done = yes;
     else call next_char;
     end;

do while (to->item.left ~= null());
     from = to->item.left;
     to->item.left = null();
     to = from;
     end;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>menu>add_tree_node.pl1, 15:55:21 Tuesday April 26, 1983
-------------------------------------------------------------------

add_tree_node: procedure (to, from, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare
     scs$allocn     entry (fix(31), pointer, area),
     from           pointer;

/*****************************************************************/

if (to->item.y > from->item.y) then do;
     to->item.left = from;                          /* temporarily */
     to->item.x = 0;                                /* permanently */
     do;
          call scs$allocn (8, new_menu, menu_ptr->menu_seg.menu_area);
          new_menu->m.first_item = to;              /* permanently */
          new_menu->m.live_item = null();           /* permanently */
          from->item.right = new_menu;              /* permanently */
          end;
     end;

else do;
     if to->item.y = from->item.y then do;
          to->item.left = from->item.left;          /* temporarily */
          from->item.left = to;                     /* permanently */
          end;
     else do; /* to->item.y < from->item.y */
          do while (to->item.y < from->item.y);
               to->item.left = from->item.left;     /* temporarily */
               from->item.left = null();            /* permanently */
               from = to->item.left;
               end;
          to->item.left = from->item.left;          /* temporarily */
          from->item.left = to;                     /* permanently */
          end;
     to->item.x = from->item.x + 1;                 /* permanently */
     end;
end;
```

```
-------------------------------------------------------------------
1   >u>nardy>menu>option_maker.pl1, 02:00:24 Thursday April 28, 1983
-------------------------------------------------------------------

option_maker: procedure (menu_ptr, text_ptr, start, finish);

%include menu_dcl;

declare

     (ioa, ioan)          entry options (variable),

     space                char(1) init (" "),
     tab                  char(1) init ("     "),
     ret                  char(1),
     ch                   char(1),

     (word_1, word_2)     char(16)vary,

     i                    fix(15),
     first_legal          fix(15),
     start                fix(15),
     finish               fix(15),

     (done, quit)         bit(1),
     yes                  bit(1) init ('1'b),
     no                   bit(1) init ('0'b),

     text_ptr             pointer,
     ascii_string         char(1) based,
     ascii_array [1:1]    char(1) based;

/*****************************************************************/

next_char: procedure;

     i = i + 1;
     ch = text_ptr->ascii_array [i];
     end;

/*****************************************************************/

next_word: procedure (word);

     declare
          word       char(16)vary,
          str_len    fix(15);

     first_legal = i;
     str_len = 0;
     do quit = no while (quit = no);
          call next_char;
          if ch = space then quit = yes;
```

```
          if ch = tab then quit = yes;
          if ch = ret then quit = yes;
          if ch = "|" then quit = yes;
          str_len = str_len + 1;
          end;
     word = substr (text_ptr->ascii_string, first_legal, str_len);
     end;

/*****************************************************************/

code_color: procedure (code);

     declare code fix(15);

     if word_2 = "black"   then code = 0;
     if word_2 = "blue"    then code = 1;
     if word_2 = "green"   then code = 2;
     if word_2 = "cyan"    then code = 3;
     if word_2 = "red"     then code = 4;
     if word_2 = "magenta" then code = 5;
     if word_2 = "yellow"  then code = 6;
     if word_2 = "white"   then code = 7;
     end;

/*****************************************************************/

/* first: set up default values, which may be overidden by user */

menu_ptr->menu_seg.leaf_prefix = '1'b;
menu_ptr->menu_seg.pop_up_mode = '1'b;
menu_ptr->menu_seg.menu_color.ov8 = 0;
menu_ptr->menu_seg.menu_color.ov9 = 0;
menu_ptr->menu_seg.menu_color.ov10 = 0;

/* now, check for explicit values */

ret = "
";
i = start;
call next_char;
do done = no while (done = no);

     /* find first non-space/tab character */
     do while ((ch = space) | (ch = tab));
          call next_char;
          end;

     /* if char is not <cr> or comment marker, then work on the line */
     if ch ~= ret then if ch ~= "|" then do;
```

```
-----------------------------------------------------------------
3    >u>nardy>menu>option_maker.pl1
-----------------------------------------------------------------


          call next_word (word_1);
          do while ((ch = space) | (ch = tab));
               call next_char;
               end;
          call next_word (word_2);

          if word_1 = "leaf_prefix" then do;
               if word_2 = "yes" then menu_ptr->menu_seg.leaf_prefix = '1'b;
               else menu_ptr->menu_seg.leaf_prefix = '0'b;
               end;

          else if word_1 = "plane_8" then
               call code_color (menu_ptr->menu_seg.menu_color.ov8);
          else if word_1 = "plane_9" then
               call code_color (menu_ptr->menu_seg.menu_color.ov9);
          else if word_1 = "plane_10" then
               call code_color (menu_ptr->menu_seg.menu_color.ov10);

          else if word_1 = "mode" then do;
               if word_2 = "pop_up" then
                    menu_ptr->menu_seg.pop_up_mode = '1'b;
               else if word_2 = "pop-up" then
                    menu_ptr->menu_seg.pop_up_mode = '1'b;
               else if word_2 = "popup" then
                    menu_ptr->menu_seg.pop_up_mode = '1'b;
               else if word_2 = "continuous" then
                    menu_ptr->menu_seg.pop_up_mode = '0'b;
               end;
          end;

     do while (ch ^= ret);
          call next_char;
          end;
     if i => finish then done = yes;
     else call next_char;
     end;

call ioa ("^rOPTIONS:");

if menu_ptr->menu_seg.leaf_prefix = '1'b then call ioa ("leaf_prefix = yes");
else call ioa ("leaf_prefix = no");

if menu_ptr->menu_seg.pop_up_mode = '1'b then call ioa ("menu mode = pop_up");
else call ioa ("menu_mode = continuous");

call ioa ("color code of plane 8  = ^i", menu_ptr->menu_seg.menu_color.ov8);
call ioa ("color code of plane 9  = ^i", menu_ptr->menu_seg.menu_color.ov9);
call ioa ("color code of plane 10 = ^i", menu_ptr->menu_seg.menu_color.ov10);
```

```
------------------------------------------------------------------
4    >u>nardy>menu>option_maker.pl1
------------------------------------------------------------------

call ioa ("(Ø=black, 1=blue, 2=green, 3=cyan, 4=red, 5=mag, 6=yel, 7=white)");
call ioa ("");

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>menu>figure_xy.pl1, Ø2:Ø4:2Ø Thursday April 28, 1983
-------------------------------------------------------------------

figure_xy: procedure (param_x, param_y, head, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare
     answer    char(1),
     askn      entry options (variable),
     ioa       entry options (variable),
     head      pointer,
     current   pointer,
     param_x   fix(15),
     param_y   fix(15),
     item_x    fix(15),
     item_y    fix(15);

item_x = param_x - 1;
item_y = param_y;

do current = head repeat current->item.left while (current ~= null());
     item_x = item_x + 1;
     if item_x > (menu_ptr->menu_seg.menu_port.x_ext /
          menu_ptr->menu_seg.item_port.x_ext) - 1 then do;
          item_x = Ø;
          item_y = item_y + 1;
          end;
     current->item.x = item_x;
     current->item.y = item_y;
     end;

do current = head repeat current->item.left while (current ~= null());
     if current->item.right ~= null() then call figure_xy
          (Ø, item_y + 1, current->item.right->m.first_item, menu_ptr);
     end;

end;
```

```
------------------------------------------------------------------
1    >u>nardy>menu>prompt_list_writer.pl1, Ø2:Ø5:29 Thursday April 28, 1983
------------------------------------------------------------------

prompt_list_writer: procedure (head_ptr);

declare

     (ioa, ioan)          entry options (variable),

     head_ptr             pointer,
     match_ptr            pointer,
     1 match              based,
          2 string        char(32)vary,
          2 data          pointer,
          2 next          pointer,

     text_ptr             pointer,
     ascii_array [1:1]    char(1) based (text_ptr),

     prompt               char(1Ø24),
     string_length        fixed(31),
     fix15                fix(15) based,
     i                    fix(15);

/*****************************************************************/

do match_ptr = head_ptr repeat match_ptr->match.next while
          (match_ptr ~= null());
     call ioa ("^r-------------------------------< match node >");
     call ioa ("                      match_ptr = ^p", match_ptr);
     call ioa ("       match_ptr->match.data = ^p", match_ptr->match.data);
     call ioa ("     match_ptr->match.string = [^a]", match_ptr->match.string);
     string_length = match_ptr->match.data->fix15;
     call ioa ("match_ptr->match.data->fix15 = ^i",
          match_ptr->match.data->fix15);
     do i = 3 to string_length + 2;
          call ioan ("^a", match_ptr->match.data->ascii_array [i]);
          end;
     call ioa ("^rmatch_ptr->match.next = ^p", match_ptr->match.next);
     end;
end;
```

```
------------------------------------------------------------------
1   >u>nardy>menu>port_writer.pl1, 02:13:07 Thursday April 28, 1983
------------------------------------------------------------------

port_writer: procedure (menu_ptr);

%include menu_dcl;

one_port: procedure (string, port_ptr);

     declare
          string         char(16)vary,
          ioa            entry options (variable),
          port_ptr       pointer,
          1 port         based (port_ptr),
               2 x_abs   fix(15),
               2 y_abs   fix(15),
               2 x_rel   fix(15),
               2 y_rel   fix(15),
               2 x_ext   fix(15),
               2 y_ext   fix(15),
               2 x_ch    fix(15),
               2 y_ch    fix(15),
               2 w_ch    fix(15),
               2 h_ch    fix(15),
               2 outline fix(15),
               2 txtmode bit(16),
               2 planes  bit(16);

     call ioa ("^r----------> ^a", string);
     call ioa ("  x_abs   = ^i", port_ptr->port.x_abs);
     call ioa ("  y_abs   = ^i", port_ptr->port.y_abs);
     call ioa ("  x_rel   = ^i", port_ptr->port.x_rel);
     call ioa ("  y_rel   = ^i", port_ptr->port.y_rel);
     call ioa ("  x_ext   = ^i", port_ptr->port.x_ext);
     call ioa ("  y_ext   = ^i", port_ptr->port.y_ext);
     call ioa ("  x_ch    = ^i", port_ptr->port.x_ch);
     call ioa ("  y_ch    = ^i", port_ptr->port.y_ch);
     call ioa ("  w_ch    = ^i", port_ptr->port.w_ch);
     call ioa ("  h_ch    = ^i", port_ptr->port.h_ch);
     call ioa ("  outline = ^i", port_ptr->port.outline);
     call ioa ("  txtmode = ^h", port_ptr->port.txtmode);
     call ioa ("  planes  = ^h", port_ptr->port.planes);
     end;

call one_port ("ground", addr (menu_ptr->menu_seg.ground_port));
call one_port ("menu",   addr (menu_ptr->menu_seg.menu_port));
call one_port ("item",   addr (menu_ptr->menu_seg.item_port));
call one_port ("title",  addr (menu_ptr->menu_seg.title_port));
call one_port ("prompt", addr (menu_ptr->menu_seg.prompt_port));
end;
```

```
------------------------------------------------------------
1   >u>nardy>menu>tree_writer.pl1, 02:15:29 Thursday April 28, 1983
------------------------------------------------------------

tree_writer: procedure (to);

%include menu_item_dcl;

declare

     (ioa, ioan)          entry options (variable),
     auxiliary            pointer,
     i                    fix(15),
     str_len              fix(15),
     fix15                fix(15) based,
     ascii_array [1:1]    char(1) based;

call ioa ("------------------------------------------------------------");
call ioa ("[item ^p] [item.left-> ^p]",
     to, to->item.left);
call ioa ("[item.right-> ^p] [item.right->m.first_item-> ^p]",
     item.right, item.right->m.first_item);
call ioa ("^ry: ^i    (x: ^i)", to->item.y, to->item.x);

call ioan ('LABEL^r"');
str_len = to->item.label->fix15;
do i = 3 to str_len + 2;
     call ioan ("^a", to->item.label->ascii_array [i]);
     end;
call ioa ('"');

call ioan ('CONTENT^r"');
if to->content ~= null() then do;
     str_len = to->item.content->fix15;
     do i = 3 to str_len + 2;
          call ioan ("^a", to->item.content->ascii_array [i]);
          end;
     end;
if to->item.content ~= null() then call ioan ('"');
call ioa ("");

if to->item.right = null() then
     call ioa ("is a LEAF node (runs a procedure)");
else call ioa ("is a MENU node (puts up a new menu)");

if to->item.right ~= null() then do;
     auxiliary = to->item.right->m.first_item;
     call tree_writer (auxiliary);
     end;
if to->item.left  ~= null() then call tree_writer (to->item.left);

end;
```

APPENDIX III

Model menu

```
------------------------------------------------------------------
1   >u>nardy>profile>typ4.menu, 02:40:05 Tuesday May 3, 1983
------------------------------------------------------------------

**MENU

typ4 font processor                    [level one]

     raw                               [raw]
          file                         [raw file]
               l dir:                       raw_dir
               init:                        init_raw
               term:                        term_raw
          draw                         [raw draw]
               one:                         one_raw_char
               all:                         all_raw_chars
          headers                      [raw headers]
               file:                        raw_file_stats
               chars:                       raw_list
          modify:                           mod_raw
          capture:                          contour

     glyph                             [glyph]
          file                         [glyph file]
               l dir:                       glyph_dir
               init old:                    init_old_glyph
               init new:                    init_new_glyph
               term:                        term_glyph
          draw                         [glyph draw]
               one:                         draw_glyph
               all:                         all_glyphs
          headers                      [glyph headers]
               file:                        glyph_file_stats
               glyphs:                      glyph_list
          modify                       [glyph mod]
               name:                        mod$glyph_name
               source:                      mod$glyph_source
               date:                        mod$glyph_date
               base:                        mod$glyph_base
               x-hgt:                       mod$glyph_xhgt
               cap:                         mod$glyph_cap
               max-hwd:                     mod$glyph_max_hwd
               last-nbr:                    mod$glyph_last_nbr
               stats:                       mod_glyph_stats
          append:                           appglyph

     chr                               [chr]
          file                         [chr file]
               l dir:                       chr_dir
               init old:                    init_old_chr
               init new:                    init_new_chr
               term:                        term_chr
          draw                         [chr draw]
```

```
------------------------------------------------------------------
2   >u>nardy>profile>typ4.menu
------------------------------------------------------------------

               one:                        draw_chr
               all:                        all_chrs
          headers                     [chr headers]
               file:                       chr_file_stats
               chrs:                       chr_list
          modify                      [chr mod]
               name:                       change$chr_name
               source:                     change$chr_source
               date:                       change$chr_date
               base:                       change$chr_base
               x-hgt:                      change$chr_xhgt
               cap:                        change$chr_cap
               max-hwd:                    change$chr_max_hwd
               last-nbr:                   change$chr_last_nbr
               stats:                      change_chr_stats
          append:                          append_char
          write:                           write_chr

     lrn:                                  lrn

     magic6:                               magic6


**PROMPTS

[level one]
select option:

raw       .raw file operations
glyph     .glyph file operations
chr       .chr file operations
lrn       list reference names
magic6    any magic_six command

[raw]
select option:

file      list, initiate, terminate
draw      draw characters ons screen
headers   list header information
modify    change header information
capture   use vidicon to digitize font

[raw file]
select option:

l dir     list .raw directory
init      initiate .raw file
term      terminate .raw file
```

```
-----------------------------------------------------------------
3   >u>nardy>profile>typ4.menu
-----------------------------------------------------------------


[raw draw]
select option:

one  draw one .raw character
all  draw all .raw characters

[raw headers]
select option:

file   list .raw file header
chars  list character headers

[glyph]
select option:

file     list, initiate, terminate
draw     draw glyphs
headers  list header information
modify   change header information
append   add .raw chars to .glyph file

[glyph file]
select option:

l dir      list .glyph directory
init old   initiate existing .glyph file
init new   create, initiate .glyph file
term       terminate .glyph file

[glyph draw]
select option:

one  draw one glyph
all  draw all glyphs

[glyph headers]
select option:

file    list .glyph file header
glyphs  list glyph headers

[glyph mod]
select option:

name      change name of glyph-set
source    change source of glyph-set
date      change date
base      change baseline
```

```
------------------------------------------------------------------
4   >u>nardy>profile>typ4.menu
------------------------------------------------------------------

x-hgt      change x-height
cap        change cap height
max-hwd    change max hgt, width, depth
last-nbr   change number of last glyph
stats      compute max-hgt -width -depth

[chr]
select option:

file      list, initiate, terminate
draw      draw chrs
headers   list header information
modify    change header information
append    add glyphs to .chr file
write     write with .chr font

[chr file]
select option:

l dir       list .chr directory
init old    initiate existing .chr file
init new    create, initiate .chr file
term        terminate .chr file

[chr draw]
select option:

one  draw one chr
all  draw all chrs

[chr headers]
select option:

file  list .chr file header
chr   list chr header information

[chr mod]
select option:

name       change name of chr-set
source     change source of chr-set
date       change date
base       change baseline
x-hgt      change x-height
cap        change cap height
max-hwd    change max hgt, width, depth
last-nbr   change number of last chr
stats      compute max-hgt -width -depth
**PORTS
```

```
-------------------------------------------------------------------
5   >u>nardy>profile>typ4.menu
-------------------------------------------------------------------

|          x    y    w    h          chx  chy  chw  chh        outl planes
ground     0    0    512  480        0    0    0    0          0    001
menu       16   16   480  224        0    0    0    0          0    001
item       0    0    80   24         7    5    9    12         1    001
title      16   448  480  32         16   1    18   24         0    001
prompt     16   256  480  192        0    150  9    16         0    001
| (chx, chy) is the starting location of type wrt its port (ergo 150)

**OPTIONS

leaf_prefix     yes         | if "yes", puts a "*" before each leaf label
                            | if "no", prints each label as is
plane_8         black       | (not used in this menu)
plane_9         black       | (not used in this menu)
plane_10        black       | color of lettering and boxes
                            | legal overlay colors:
                            | black, red, blue, green,
                            | cyan, yellow, magenta, white

mode            pop_up      | choice: pop_up (=default), or continuous
```

APPENDIX IV

Source code for use package

```
------------------------------------------------------------------
1   >u>nardy>use>use.bind, 02:18:10 Thursday April 28, 1983
------------------------------------------------------------------

use
;
;                    menu_manager calls all other procedures
;                    menu_manager contains use: entry;
menu_manager
;
;
draw_item
erase_item
draw_title
erase_title
draw_prompt
erase_prompt
draw_menu
erase_menu
erase_menu_planes
kill_all_living_things
kill_menu_list
restore_menu
before_dollar
match_finder
init_menu
```

```
------------------------------------------------------------------
1   >u>nardy>menu>menu_dcl.incl.pl1, 02:53:49 Thursday April 28, 1983
------------------------------------------------------------------

declare

     menu_ptr             pointer,
     1 menu_seg           based (menu_ptr),
          2 leaf_prefix   bit(1),
          2 pop_up_mode   bit(1),
          2 menu_color,
               3 ov8      fix(15),
               3 ov9      fix(15),
               3 ov10     fix(15),
          2 ground_port,
               3 x_abs    fix(15),
               3 y_abs    fix(15),
               3 x_rel    fix(15),
               3 y_rel    fix(15),
               3 x_ext    fix(15),
               3 y_ext    fix(15),
               3 x_ch     fix(15),
               3 y_ch     fix(15),
               3 w_ch     fix(15),
               3 h_ch     fix(15),
               3 outline  fix(15),
               3 txtmode  bit(16),
               3 planes   bit(16),
          2 menu_port     like ground_port,
          2 item_port     like ground_port,
          2 title_port    like ground_port,
          2 prompt_port   like ground_port,
          2 root_data     pointer,
          2 root_menu     pointer,
          2 menu_area     area (16000);
```

```
-----------------------------------------------------------------
1    >u>nardy>menu>menu_item_dcl.incl.pl1, 02:54:42 Thursday April 28, 1983
-----------------------------------------------------------------

declare

     new_menu            pointer,
     1 m                 based (new_menu),
          2 first_item   pointer,
          2 live_item    pointer,

     to                  pointer,
     1 item              based (to),
          2 label        pointer,
          2 content      pointer,
          2 right        pointer,
          2 left         pointer,
          2 x            fix(15),
          2 y            fix(15);
```

```
------------------------------------------------------------------
1   >u>nardy>use>menu_manager.pl1, 02:21:16 Thursday April 28, 1983
------------------------------------------------------------------

menu_manager: procedure;

use: entry ();

%include menu_dcl;
%include menu_item_dcl;
%include map_dcl;

declare

     (ioa, ioan)          entry options (variable),
     askn                 entry options (variable),
     iocs$getc            entry (char(1)),
     scs$cl               entry options (variable),
     scs$get_arg_count    entry (fix(15)),
     scs$get_arg_info     entry (fix(15), bit(16), fix(15), pointer),
     scs$expand_path      entry (char(168)vary, char(168)vary, char(32)vary,
                               fix(31)),
     hcs$terminate        entry (pointer),
     tmr                  entry options (variable),

     grin$ocolor          entry (fix, fix, fix),

     $init_menu           entry (char(168)vary, char(32)vary, pointer),
     $restore_menu        entry (pointer),
     $erase_menu_planes   entry (pointer),
     $kill_all_living_things
                          entry (pointer),
     $draw_title          entry (pointer, bit(1), pointer),
     $draw_prompt         entry (pointer, bit(1), pointer),
     $erase_prompt        entry (pointer),
     $draw_menu           entry (pointer, pointer),
     $match_finder        entry (pointer, fix(15), fix(15),
                               pointer, pointer, pointer),
     $before_dollar       entry (char(32)vary) returns (char(32)vary);

declare
     num_args             fix(15),
     arg_length           fix(15),
     arg_type             bit(16),
     arg_ptr              pointer,
     arg_string           char(32) based (arg_ptr),

     dir_name             char(168)vary,
     entry_name           char(32)vary,
     rec_name             char(32)vary,
     menu_name            char(32)vary,
     garbage_name         char(32)vary,
     program_name         char(32)vary,
```

```
------------------------------------------------------------------
2   >u>nardy>use>menu_manager.pll
------------------------------------------------------------------

     answer              char(1),
     ascii_array [1:1]   char(1) based,

     temp_y_float        float(23),
     temp_x_float        float(23),
     x_flt_items         float(23),
     y_flt_items         float(23),
     (fp_x, fp_y)        float(23),

     x_num_items         fix(15),
     y_num_items         fix(15),
     x_num_pixels        fix(15),
     y_num_pixels        fix(15),
     y_tab_extent        fix(15),
     (scrn_x, scrn_y)    fix(15),
     (i, x, y, z)        fix(15),
     str_len             fix(15),
     fix15_num           fix(15) based,

     syscode             fix(31),
     oops                condition,
     systemao            condition,
     break               condition,

     quitp               bit(1),
     legalp              bit(1),
     no                  bit(1) init ('0'b),
     yes                 bit(1) init ('1'b),
     normal              bit(1) init ('0'b),
     inverse             bit(1) init ('1'b),

     match_item          pointer,
     match_header        pointer,
     current             pointer;

/*****************************************************************/

on systemao begin;
     end;

on oops begin;
     goto exit;
     end;

on break begin;
     call ioan ("^r--> confirm BREAK (y/n) ");
     call iocs$getc (answer);
     call ioa ("^r");
     if answer = "y" then goto exit;
```

```
-----------------------------------------------------------------
3    >u>nardy>use>menu_manager.pl1
-----------------------------------------------------------------

     end;

call scs$get_arg_count (num_args);
if num_args = 0 then call askn ("name of menu: ", menu_name);
else do;
     call scs$get_arg_info (1, arg_type, arg_length, arg_ptr);
     menu_name = substr (arg_ptr->arg_string, 1, arg_length);
     end;
unspec (menu_ptr) = '000c0000'b4;
call scs$expand_path (menu_name, dir_name, garbage_name, syscode);
call $init_menu (dir_name, menu_name, menu_ptr);

call grin$ocolor (menu_ptr->menu_seg.menu_color.ov8,
                  menu_ptr->menu_seg.menu_color.ov9,
                  menu_ptr->menu_seg.menu_color.ov10);

call map$setup;

call ioan ("^r--> resume or fresh start? (r/s) ");
call iocs$getc (answer);
call ioa ("");
if answer = "r" then call $restore_menu (menu_ptr);
else do;
     current = menu_ptr->menu_seg.root_menu->m.first_item;
     menu_ptr->menu_seg.root_menu->m.live_item = current;
     call $kill_all_living_things (current->item.right);
     call $draw_title  (current, inverse, menu_ptr);
     call $draw_prompt (current, normal, menu_ptr);
     call $draw_menu   (current->item.right, menu_ptr);
     end;

x_num_items =
     menu_ptr->menu_seg.menu_port.x_ext / menu_ptr->menu_seg.item_port.x_ext;
x_flt_items = x_num_items;
y_num_items =
     menu_ptr->menu_seg.menu_port.y_ext / menu_ptr->menu_seg.item_port.y_ext;
y_flt_items = y_num_items;
x_num_pixels = menu_ptr->menu_seg.item_port.x_ext * x_num_items;
y_num_pixels = menu_ptr->menu_seg.item_port.y_ext * y_num_items;
temp_x_float = x_num_pixels;
temp_y_float = y_num_pixels;
y_tab_extent = (temp_y_float * 2000.0) / temp_x_float;

do quitp = no while (quitp = no);
     call map$tablet (77, 601, 2048, y_tab_extent);
     call map$window (0, 0, x_flt_items, y_flt_items);
     call map$port (menu_ptr->menu_seg.menu_port.x_abs,
          menu_ptr->menu_seg.menu_port.y_abs, x_num_pixels, y_num_pixels);
     do legalp = no while (legalp = no);
```

```
-----------------------------------------------------------------
4   >u>nardy>use>menu_manager.pl1
-----------------------------------------------------------------

          do z = 0 while (z < 1);
               call map$fp (scrn_x, scrn_y, fp_x, fp_y, z);
               end;
          do while (z ~= 0);
               call map$fp (scrn_x, scrn_y, fp_x, fp_y, z);
               end;
          x = fp_x;
          y = fp_y;
          call $match_finder (menu_ptr->menu_seg.root_menu,
               x, y, match_item, match_header, menu_ptr);
          if match_item ~= null() then legalp = yes;
          end;

     if quitp = no then do;
          call $draw_title (match_item, inverse, menu_ptr);
          call $erase_prompt (menu_ptr);
          if match_item->item.right = null() then do;
               /* you are about to call a program-- now, you might break
               out in the middle of it, so you have to prepare the menu
               tree for that eventuality: first update title, then get
               program_name, then set live_item pointed to by match_header
               to null(): this sets you up for restoring the menu to
               how it was before you picked the last item */
               program_name = "";
               str_len = match_item->item.content->fix15_num;
               do i = 3 to str_len + 2;
                    program_name = program_name ||
                         match_item->item.content->ascii_array [i];
               end;
               /* anticipating successful completion of program: */
               match_header->m.live_item = null();
               if menu_ptr->menu_seg.pop_up_mode = '1'b
                    then call $erase_menu_planes (menu_ptr);
               call scs$cl (program_name);
               call tmr ($before_dollar(program_name));
               if menu_ptr->menu_seg.pop_up_mode = '1'b
                    then call $erase_menu_planes (menu_ptr);
               call grin$ocolor (menu_ptr->menu_seg.menu_color.ov8,
                    menu_ptr->menu_seg.menu_color.ov9,
                    menu_ptr->menu_seg.menu_color.ov10);
               call $restore_menu (menu_ptr); /* makes match_item normal */
               end;
          else do;
               call $draw_prompt (match_item, normal, menu_ptr);
               call $draw_menu
                    (match_header->m.live_item->item.right, menu_ptr);
               end;
          end;
     end;
```

```
------------------------------------------------------------------
5    >u>nardy>use>menu_manager.pll
------------------------------------------------------------------


exit:

call $erase_menu_planes (menu_ptr);
call hcs$terminate (menu_ptr);
call tmr ("map_param_01");

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>draw_item.pl1, 02:25:38 Thursday April 28, 1983
-------------------------------------------------------------------

draw_item: procedure (instance, norm_inv, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     grin$rectv          entry (fix, fix, fix, fix, bit(16), fix),
     grin$text           entry (fix, fix, bit(16), char(1024)vary),

     instance            pointer,

     (x1, y1, x2, y2)    fix(15),
     (i, x, y)           fix(15),
     right_margin        fix(15),
     bottom_margin       fix(15),
     off                 fix(15) init (0),
     on                  fix(15) init (1),
     fix15               fix(15) based,

     text_mode           bit(16),
     norm_inv            bit(1),   /* normal = '0'b, inverse = '1'b */

     ascii_array [1:1]   char(1) based,
     ret                 char(1),
     one_char            char(1),
     ascii_num           fix(7) defined one_char;

/*****************************************************************/

/* first, do the box */

x1 = menu_ptr->menu_seg.item_port.x_abs +
     (instance->item.x * menu_ptr->menu_seg.item_port.x_ext);
y1 = menu_ptr->menu_seg.item_port.y_abs +
     (instance->item.y * menu_ptr->menu_seg.item_port.y_ext);
x2 = x1 + menu_ptr->menu_seg.item_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.item_port.y_ext - 2;

x = x1 + menu_ptr->menu_seg.item_port.x_ch;
y = y1 + menu_ptr->menu_seg.item_port.y_ch;

call grin$rectv (x1, y1, x2, y2, menu_ptr->menu_seg.item_port.planes, on);

if norm_inv = '0'b then do;  /* normal */
     x1 = x1 + menu_ptr->menu_seg.item_port.outline;
     y1 = y1 + menu_ptr->menu_seg.item_port.outline;
     x2 = x2 - menu_ptr->menu_seg.item_port.outline;
     y2 = y2 - menu_ptr->menu_seg.item_port.outline;
```

```
------------------------------------------------------------------
2   >u>nardy>use>draw_item.pl1
------------------------------------------------------------------

     call grin$rectv
          (x1, y1, x2, y2, menu_ptr->menu_seg.item_port.planes, off);
     end;

right_margin  = x2;
bottom_margin = y1;

/* next, do the characters */

ret = "
";

text_mode = menu_ptr->menu_seg.item_port.txtmode;
substr (text_mode, 1, 1) = norm_inv;

if menu_ptr->menu_seg.leaf_prefix = '1'b then
     if instance->item.right = null() then do;
          call grin$text (x, y, text_mode, "*");
          x = x + menu_ptr->menu_seg.item_port.w_ch;
          end;

do i = 3 to instance->item.label->fix15 + 2;
     one_char = instance->item.label->ascii_array [i];
     if ((ascii_num > 95) & (ascii_num < 127)) then
          ascii_num = ascii_num - 32;
     if one_char = " " then x = x + menu_ptr->menu_seg.item_port.w_ch;
     else if one_char = "     " then  /* sleazy-tab */
          x = x + 5 * menu_ptr->menu_seg.item_port.w_ch;
     else if one_char = ret then do;
          x = menu_ptr->menu_seg.item_port.x_abs +
               menu_ptr->menu_seg.item_port.x_ch;
          y = y - menu_ptr->menu_seg.item_port.h_ch;
          end;
     else do;
          if (x + menu_ptr->menu_seg.item_port.w_ch < right_margin) then
               if (y > bottom_margin) then do;
                    call grin$text (x, y, text_mode, one_char);
                    x = x + menu_ptr->menu_seg.item_port.w_ch;
                    end;
          end;
     end;
end;
```

```
------------------------------------------------------------------
1   >u>nardy>use>erase_item.pl1, 02:27:53 Thursday April 28, 1983
------------------------------------------------------------------

erase_item: procedure (instance, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     grin$rectv          entry (fix, fix, fix, fix, bit(16), fix),

     instance            pointer,

     (x1, y1, x2, y2)    fix(15),
     off                 fix(15) init (0),
     on                  fix(15) init (1);

/*****************************************************************/

x1 = menu_ptr->menu_seg.item_port.x_abs +
     (instance->item.x * menu_ptr->menu_seg.item_port.x_ext);
y1 = menu_ptr->menu_seg.item_port.y_abs +
     (instance->item.y * menu_ptr->menu_seg.item_port.y_ext);
x2 = x1 + menu_ptr->menu_seg.item_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.item_port.y_ext - 2;

call grin$rectv (x1, y1, x2, y2, menu_ptr->menu_seg.item_port.planes, off);

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>draw_title.pl1, 02:28:39 Thursday April 28, 1983
-------------------------------------------------------------------

draw_title: procedure (instance, norm_inv, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     ioa                  entry options (variable),
     grin$rectv           entry (fix, fix, fix, fix, bit(16), fix),
     grin$text            entry (fix, fix, bit(16), char(1024)vary),

     instance             pointer,

     (x1, y1, x2, y2)     fix(15),
     (i, x, y)            fix(15),
     right_margin         fix(15),
     bottom_margin        fix(15),
     off                  fix(15) init (0),
     on                   fix(15) init (1),
     fix15                fix(15) based,

     text_mode            bit(16),
     norm_inv             bit(1),   /* normal = '0'b, inverse = '1'b */

     ascii_array [1:1]    char(1) based,
     ret                  char(1),
     one_char             char(1),
     ascii_num            fix(7) defined one_char;

/*****************************************************************/

/* first, do the box */

x1 = menu_ptr->menu_seg.title_port.x_abs;
y1 = menu_ptr->menu_seg.title_port.y_abs;
x2 = x1 + menu_ptr->menu_seg.title_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.title_port.y_ext - 2;

x = x1 + menu_ptr->menu_seg.title_port.x_ch;
y = y1 + menu_ptr->menu_seg.title_port.y_ch;

call grin$rectv (x1, y1, x2, y2, menu_ptr->menu_seg.title_port.planes, on);

if norm_inv = '0'b then do;  /* normal */
     x1 = x1 + menu_ptr->menu_seg.title_port.outline;
     y1 = y1 + menu_ptr->menu_seg.title_port.outline;
     x2 = x2 - menu_ptr->menu_seg.title_port.outline;
     y2 = y2 - menu_ptr->menu_seg.title_port.outline;
     call grin$rectv
```

```
------------------------------------------------------------------------
2    >u>nardy>use>draw_title.pl1
------------------------------------------------------------------------

          (x1, y1, x2, y2, menu_ptr->menu_seg.title_port.planes, off);
     end;

right_margin  = x2;
bottom_margin = y1;

/* next, do the characters */

ret = "
";

text_mode = menu_ptr->menu_seg.title_port.txtmode;
substr (text_mode, 1, 1) = norm_inv;

if menu_ptr->menu_seg.leaf_prefix = '1'b then
     if instance->item.right = null() then do;
          call grin$text (x, y, text_mode, "*");
          x = x + menu_ptr->menu_seg.title_port.w_ch;
          end;

do i = 3 to instance->item.label->fix15 + 2;
     one_char = instance->item.label->ascii_array [i];
     if ^((ascii_num > 95) & (ascii_num < 127)) then
          ascii_num = ascii_num - 32;
     if one_char = " " then x = x + menu_ptr->menu_seg.title_port.w_ch;
     else if one_char = "     " then  /* sleazy-tab */
          x = x + 5 * menu_ptr->menu_seg.title_port.w_ch;
     else if one_char = ret then do;
          x = menu_ptr->menu_seg.title_port.x_abs +
               menu_ptr->menu_seg.title_port.x_ch;
          y = y - menu_ptr->menu_seg.title_port.h_ch;
          end;
     else do;
          if (x + menu_ptr->menu_seg.title_port.w_ch < right_margin) then
               if (y > bottom_margin) then do;
                    call grin$text (x, y, text_mode, one_char);
                    x = x + menu_ptr->menu_seg.title_port.w_ch;
                    end;
          end;
     end;
end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>erase_title.pl1, 02:30:32 Thursday April 28, 1983
-------------------------------------------------------------------

erase_title: procedure (menu_ptr);

%include menu_dcl;

declare

     grin$rectv          entry (fix, fix, fix, fix, bit(16), fix),

     (x1, y1, x2, y2)    fix(15),
     off                 fix(15) init (0),
     on                  fix(15) init (1);

/*****************************************************************/

x1 = menu_ptr->menu_seg.title_port.x_abs;
y1 = menu_ptr->menu_seg.title_port.y_abs;
x2 = x1 + menu_ptr->menu_seg.title_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.title_port.y_ext - 2;

call grin$rectv (x1, y1, x2, y2, menu_ptr->menu_seg.title_port.planes, off);

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>draw_prompt.pl1, 02:31:02 Thursday April 28, 1983
-------------------------------------------------------------------

draw_prompt: procedure (instance, norm_inv, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     ioa                  entry options (variable),
     grin$rectv           entry (fix, fix, fix, fix, bit(16), fix),
     grin$text            entry (fix, fix, bit(16), char(1024)vary),

     instance             pointer,

     (x1, y1, x2, y2)     fix(15),
     (i, x, y)            fix(15),
     right_margin         fix(15),
     bottom_margin        fix(15),
     off                  fix(15) init (0),
     on                   fix(15) init (1),
     fix15                fix(15) based,

     text_mode            bit(16),
     norm_inv             bit(1),   /* normal = '0'b, inverse = '1'b */

     ascii_array [1:1]    char(1) based,
     ret                  char(1),
     one_char             char(1),
     ascii_num            fix(7) defined one_char;

/*****************************************************************/

/* first, do the box */

x1 = menu_ptr->menu_seg.prompt_port.x_abs;
y1 = menu_ptr->menu_seg.prompt_port.y_abs;
x2 = x1 + menu_ptr->menu_seg.prompt_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.prompt_port.y_ext - 2;

x = x1 + menu_ptr->menu_seg.prompt_port.x_ch;
y = y1 + menu_ptr->menu_seg.prompt_port.y_ch;
/*
call grin$rectv (x1, y1, x2, y2, menu_ptr->menu_seg.prompt_port.planes, on);
*/
if norm_inv = '0'b then do;  /* normal */
     x1 = x1 + menu_ptr->menu_seg.prompt_port.outline;
     y1 = y1 + menu_ptr->menu_seg.prompt_port.outline;
     x2 = x2 - menu_ptr->menu_seg.prompt_port.outline;
     y2 = y2 - menu_ptr->menu_seg.prompt_port.outline;
     call grin$rectv
```

```
-----------------------------------------------------------------
2   >u>nardy>use>draw_prompt.pll
-----------------------------------------------------------------

          (x1, y1, x2, y2, menu_ptr->menu_seg.prompt_port.planes, off);
     end;

right_margin  = x2;
bottom_margin = y1;

/* next, do the characters */

ret = "
";

text_mode = menu_ptr->menu_seg.prompt_port.txtmode;
substr (text_mode, 1, 1) = norm_inv;

do i = 3 to instance->item.content->fix15 + 2;
     one_char = instance->item.content->ascii_array [i];
     if ((ascii_num > 95) & (ascii_num < 127)) then
          ascii_num = ascii_num - 32;
     if one_char = " " then x = x + menu_ptr->menu_seg.prompt_port.w_ch;
     else if one_char = "     " then  /* sleazy-tab */
          x = x + 5 * menu_ptr->menu_seg.prompt_port.w_ch;
     else if one_char = ret then do;
          x = menu_ptr->menu_seg.prompt_port.x_abs +
               menu_ptr->menu_seg.prompt_port.x_ch;
          y = y - menu_ptr->menu_seg.prompt_port.h_ch;
          end;
     else do;
          if (x + menu_ptr->menu_seg.prompt_port.w_ch < right_margin) then
               if (y > bottom_margin) then do;
                    call grin$text (x, y, text_mode, one_char);
                    x = x + menu_ptr->menu_seg.prompt_port.w_ch;
                    end;
          end;
     end;
end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>erase_prompt.pl1, Ø2:32:51 Thursday April 28, 1983
-------------------------------------------------------------------

erase_prompt: procedure (menu_ptr);

%include menu_dcl

declare
     grin$rectv          entry (fix, fix, fix, fix, bit(16), fix),
     (x1, y1, x2, y2)    fix(15),
     off                 fix(15) init (Ø);

/*****************************************************************/

x1 = menu_ptr->menu_seg.prompt_port.x_abs;
y1 = menu_ptr->menu_seg.prompt_port.y_abs;
x2 = x1 + menu_ptr->menu_seg.prompt_port.x_ext - 2;
y2 = y1 + menu_ptr->menu_seg.prompt_port.y_ext - 2;

call grin$rectv
     (x1, y1, x2, y2, menu_ptr->menu_seg.prompt_port.planes, off);

end;
```

```
-------------------------------------------------------------------
1    >u>nardy>use>draw_menu.pl1, 02:33:45 Thursday April 28, 1983
-------------------------------------------------------------------

draw_menu: procedure (new_menu, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare
     $draw_item     entry (pointer, bit(1), pointer),
     item_ptr       pointer,
     normal         bit(1) init ('0'b),
     inverse        bit(1) init ('1'b);

do item_ptr = new_menu->m.first_item
     repeat item_ptr->item.left while (item_ptr ^= null());

     if item_ptr = new_menu->m.live_item then
          call $draw_item (item_ptr, inverse, menu_ptr);
     else call $draw_item (item_ptr, normal,  menu_ptr);
     end;

end;
```

```
-----------------------------------------------------------------
1   >u>nardy>use>erase_menu.pl1, Ø2:34:52 Thursday April 28, 1983
-----------------------------------------------------------------

erase_menu: procedure (new_menu, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare
     $erase_item    entry (pointer, pointer),
     item_ptr       pointer,
     normal         bit(1) init ('Ø'b),
     inverse        bit(1) init ('1'b);

do item_ptr = new_menu->m.first_item
     repeat item_ptr->item.left while (item_ptr ~= null());

     call $erase_item (item_ptr,  menu_ptr);
     end;

end;
```

```
-------------------------------------------------------------
1   >u>nardy>use>erase_menu_planes.pl1, 02:42:15 Thursday April 28, 1983
-------------------------------------------------------------

erase_menu_planes: procedure (menu_ptr);

%include menu_dcl

check: procedure (pos, bit_str);
     declare
          grin$rectv entry (fix, fix, fix, fix, bit(16), fix),
          off       fix(15) init (0),
          pos       fix(15),
          bit_str   bit(16);

     if substr (menu_ptr->menu_seg.ground_port.planes, pos, 1) = '1'b
          then call grin$rectv (0, 0, 511, 511, bit_str, off);
     else if substr (menu_ptr->menu_seg.menu_port.planes, pos, 1) = '1'b
          then call grin$rectv (0, 0, 511, 511, bit_str, off);
     else if substr (menu_ptr->menu_seg.item_port.planes, pos, 1) = '1'b
          then call grin$rectv (0, 0, 511, 511, bit_str, off);
     else if substr (menu_ptr->menu_seg.title_port.planes, pos, 1) = '1'b
          then call grin$rectv (0, 0, 511, 511, bit_str, off);
     else if substr (menu_ptr->menu_seg.prompt_port.planes, pos, 1) = '1'b
          then call grin$rectv (0, 0, 511, 511, bit_str, off);
     end;

/*****************************************************************/

call check (6, '0400'b4);
call check (7, '0200'b4);
call check (8, '0100'b4);

end;
```

```
-----------------------------------------------------------------
1   >u>nardy>use>kill_all_living_things.pl1, 02:35:20 Thursday April 28, 19
-----------------------------------------------------------------

kill_all_living_things: procedure (new_menu);

%include menu_item_dcl;

declare item_ptr pointer;

if new_menu->m.live_item ^= null() then new_menu->m.live_item = null();

do item_ptr = new_menu->m.first_item repeat item_ptr->item.left
     while (item_ptr ^= null());

     if item_ptr->item.right ^= null()
          then call kill_all_living_things (item_ptr->item.right);
     end;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>use>kill_menu_list.pl1, 02:36:07 Thursday April 28, 1983
-------------------------------------------------------------------

kill_menu_list: procedure (new_menu, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare
     next_menu      pointer,
     $erase_menu    entry (pointer, pointer);

call $erase_menu (new_menu, menu_ptr);

if new_menu->m.live_item ^= null() then do;
     if new_menu->m.live_item->item.right ^= null() then do;
          next_menu = new_menu->m.live_item->item.right;
          call kill_menu_list (next_menu, menu_ptr);
          new_menu->m.live_item = null();
          end;
     end;
end;
```

```
-------------------------------------------------------------------
1    >u>hardy>use>restore_menu.pl1, 02:40:14 Thursday April 28, 1983
-------------------------------------------------------------------

restore_menu: procedure (menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     $draw_menu      entry (pointer, pointer),
     $draw_title     entry (pointer, bit(1), pointer),
     $draw_prompt    entry (pointer, bit(1), pointer),

     current         pointer,
     previous        pointer,
     quitp           bit(1),
     no              bit(1) init ('0'b),
     yes             bit(1) init ('1'b),
     normal          bit(1) init ('0'b),
     inverse         bit(1) init ('1'b);

/*****************************************************************/

current = menu_ptr->menu_seg.root_menu->m.first_item;
menu_ptr->menu_seg.root_menu->m.live_item = current;

/*
current points at the live item on a menu
at the outset current points to the root node--the one that
     contains the title and points to the first level--of the menu
do this loop while (the live item points to another menu)
     call draw_menu
          (header pointed to by the part of current which points to next men
     current = the live item which is pointed to by the header which is
          pointed to by the "right" part of the current live item
     a case analysis here:
          you have derived the next pointer, but it may be meaningless
          if the live item pointed to by the header is a null item
          or if the live item points to a null header
          then quit the loop
     end
*/

do quitp = no while (quitp = no);
     call $draw_menu (current->item.right, menu_ptr);
     previous = current;
     current = current->item.right->m.live_item;
     if current = null() then quitp = yes;
     else if current->item.right = null() then quitp = yes;
     end;
call $draw_title  (previous, inverse, menu_ptr);
```

```
-------------------------------------------------------------------
2    >u>nardy>use>restore_menu.pll
-------------------------------------------------------------------

call $draw_prompt (previous, normal, menu_ptr);

end;
```

```
-------------------------------------------------------------------
1    >u>nardy>use>before_dollar.pl1, Ø2:38:19 Thursday April 28, 1983
-------------------------------------------------------------------

before_dollar: procedure (program_name) returns (char(32)vary);

declare
     i              fix(15),
     max_len        fix(15),
     program_name   char(32)vary,
     sub_string     char(32)vary;

max_len = length (program_name);
sub_string = "";

do i = 1 to max_len;
     if substr (program_name, i, 1) = "$" then do;
          if i > 1 then return (sub_string);
          else sub_string = "$";
          end;
     else sub_string = sub_string || substr (program_name, i, 1);
     end;

return (program_name);

end;
```

```
-------------------------------------------------------------------
1    >u>nardy>use>match_finder.pl1, 02:46:06 Thursday April 28, 1983
-------------------------------------------------------------------

match_finder: procedure
     (new_menu, match_x, match_y, match_item, match_header, menu_ptr);

%include menu_dcl;
%include menu_item_dcl;

declare

     $draw_item      entry (pointer, bit(1), pointer),
     $draw_menu      entry (pointer, pointer),
     $kill_menu_list
                     entry (pointer, pointer),

     match_x         fix(15),
     match_y         fix(15),
     item_ptr        pointer,
     match_item      pointer,
     match_header    pointer,

     endp            bit(1),
     no              bit(1) init ('0'b),
     yes             bit(1) init ('1'b),
     inverse         bit(1) init ('1'b);

/*****************************************************************/

match_item = null();
item_ptr = new_menu->m.first_item;
do endp = no while (endp = no);

     if (item_ptr->item.x = match_x) & (item_ptr->item.y = match_y) then do;
          match_item = item_ptr;
          endp = yes;
          end;
     else do;
          item_ptr = item_ptr->item.left;
          if item_ptr = null() then endp = yes;
          end;
     end;

if match_item = null() then do;
     if new_menu->m.live_item ^= null() then do;
          call match_finder (new_menu->m.live_item->item.right,
               match_x, match_y, match_item, match_header, menu_ptr);
          end;
     end;


else do; /* you have a match item, now check for level */
```

```
-------------------------------------------------------------------
2    >u>nardy>use>match_finder.pl1
-------------------------------------------------------------------


     if new_menu->m.live_item ^= null() then do;
          call $kill_menu_list (new_menu, menu_ptr);
          end;
     match_header = new_menu;
     match_header->m.live_item = match_item;
     call $draw_menu (new_menu, menu_ptr);
     end;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>use>init_menu.pl1, Ø2:43:3Ø Thursday April 28, 1983
------------------------------------------------------------------

init_menu: procedure (dir_name, menu_file, menu_ptr);

declare
     ioa             entry options (variable),
     com_error       entry options (variable),
     hcs$initiate_w_options entry (char(168)vary, char(32)vary,
                     char(32)vary, bit(1), pointer, fix(31)),
     dir_name        char(168)vary,
     menu_file       char( 32)vary,
     menu_ptr        pointer,
     syscode         fix(31),
     oops            condition;

syscode_manager: procedure (string);
     declare string char(64)vary;
     call com_error (syscode, "init_menu", string);
     signal oops;
     end;

on oops begin;
     goto exit;
     end;

call hcs$initiate_w_options
     (dir_name, menu_file, menu_file, '1'b, menu_ptr, syscode);
if syscode < Ø then call syscode_manager ("initiating " || menu_file);
else call ioa ("--> menu '^a' has been initiated", menu_file);

exit:;
end;
```

APPENDIX V

Source code for contour package

```
-------------------------------------------------------------------
1   >u>nardy>edge>contour.bind, 03:32:11 Thursday April 28, 1983
-------------------------------------------------------------------

contour
;
c5
step5
summary5
review
vidicon
alignment
tablet
io_util
```

```
-------------------------------------------------------------------
1    >u>nardy>edge>raw_file_structure.incl.pl1, 03:32:34 Thursday April 28, 19
-------------------------------------------------------------------

declare
     raw                 pointer,
     1 font              based (raw),
          2 name         char(32)vary,
          2 source       char(32)vary,
          2 date         char(32)vary,
          2 cap_line     fix(15),
          2 x_line       fix(15),
          2 base_line    fix(15),
          2 serial_num   fix(15),
          2 chr [1:128],
               3 name    char(12)vary,
               3 index   fix(15),
          2 vector [1:1] bit(16);
```

```
------------------------------------------------------------------
1   >u>nardy>edge>c5.pl1, 03:39:11 Thursday April 28, 1983
------------------------------------------------------------------

c5: procedure;

%include raw_file_structure;

declare
     (ioa, ioan)     entry options (variable),
     askn            entry options (variable),
     iocs$get        entry (char(1)vary),
     grin$ocolor     entry (fix, fix, fix),
     grin$linev      entry (fix, fix, fix, fix, bit(16), fix),
     grin$rectv      entry (fix, fix, fix, fix, bit(16), fix),
     hcs$append_seg  entry (char(168)vary, char(32)vary, fix(31)),
     hcs$initiate_w_options entry (char(168)vary, char(32)vary,
                     char(32)vary, bit(1), pointer, fix(31)),
     hcs$terminate   entry (pointer),

     $vidicon        entry,
     $tablet         entry (fix(15), fix(15), fix(15)),
     $step5          entry (fix, fix, fix, fix, fix, pointer),
     $review         entry (pointer),
     $alignment      entry (pointer);

declare
     (x, y, z)            fix(15),
     (xorg, yorg)         fix(15),
     (xstep, ystep)       fix(15),
     on                   fix(15) init (1),
     off                  fix(15) init (0);

declare
     instruction          fix(15),
     open_sequence        fix(15) init (1),
     close_sequence       fix(15) init (2),
     open_contour         fix(15) init (3),
     close_forward        fix(15) init (4),
     step_forward         fix(15) init (5),
     close_fwd_right      fix(15) init (6),
     step_fwd_right       fix(15) init (7),
     rotate_cc            fix(15) init (8),
     check_again          fix(15) init (9),
     look_for_edge        fix(15) init (10),
     keep_looking         fix(15) init (11),
     bad_start            fix(15) init (12),
     close_file           fix(15) init (13);

declare
     planes8910      bit(16) init ('0700'b4),
     plane8          bit(16) init ('0100'b4),
     plane10         bit(16) init ('0400'b4),
```

```
------------------------------------------------------------------
2   >u>nardy>edge>c5.pll
------------------------------------------------------------------

     blue            fix(15) init (1),
     green           fix(15) init (2),
     white           fix(15) init (7),
     font_file       char(32)vary,
     answer          char(3)vary,
     yes             char(3)vary init ("yes"),
     syscode         fix(31),
     systemao        condition;

/*****************************************************************/

check_ahead: procedure (tentx, tenty, xorg, yorg, instruction);

     declare
          (tentx, tenty) fix(15),
          (xorg, yorg)   fix(15),
          instruction    fix(15);

     declare
          grin$read      entry (fix, fix, fix, fix, bit(32)),
          grin$uncolor   entry (fix, fix, fix, bit(32)),
          (r, g, b)      fix(15),
          pixel          bit(32);

     call grin$read    (tentx, tenty, tentx, tenty, pixel);
     call grin$uncolor (b, g, r, pixel);
     if (r + b + g) > 383 then do; /* in: check for origin match */
          if instruction = check_again then do;
               if (tentx = xorg) & (tenty = yorg) then
                    instruction = close_fwd_right;
               else instruction = step_fwd_right;
               end;
          else if (tentx = xorg) & (tenty = yorg) then
               instruction = close_forward;
          else if instruction = look_for_edge then
               instruction = bad_start;
          else if instruction = keep_looking then instruction = open_contour;
          else instruction = check_again;
          end;
     else do; /* out: no need to check for origin match */
          if instruction = check_again then instruction = step_forward;
          else if instruction = look_for_edge then
               instruction = keep_looking;
          else if instruction = keep_looking then instruction = keep_looking;
          else instruction = rotate_cc;
          end;
     end;

 *****************************************************************/
```

```
-------------------------------------------------------------------
3   >u>nardy>edge>c5.pl1
-------------------------------------------------------------------


on systemao begin;
     end;

contour: entry;

call ioa  ("^ocsfont capture program");
call askn ("name of font_file (no spaces: use_underbars): ", font_file);
if length (font_file) > 28 then font_file = substr (font_file, 1, 28);
font_file = font_file || ".raw";
call hcs$append_seg (">u>type>raw", font_file, syscode);
unspec (raw) = '000090000'b4;
call hcs$initiate_w_options
     (">u>type>raw", font_file, font_file, '1'b, raw, syscode);
call ioa ("--> font_file '^a' appended and initiated", font_file);
raw->font.name = font_file;
call askn ("source of font (no spaces: use_underbars): ", raw->font.source);
call askn ("today's date (no spaces: use_underbars):   ", raw->font.date);

raw->font.serial_num = 1;
raw->font.chr[1].index = 1;

call grin$ocolor (blue, green, white);
call grin$rectv (0, 0, 511, 511, planes8910, off);
call $alignment (raw);
call $vidicon;

call $step5 (0, 0, xstep, ystep, open_sequence, raw);
do while ('1'b);
     do z = 0 while (z < 1);
          call $tablet (xorg, yorg, z);
          if z = 2 then do;
               do z = 2 while (z = 2);
                    call $tablet (x, y, z);
                    end;
               call ioan ("--> BEGIN NEW CHARACTER....");
               call grin$rectv (0, 0, 511, 511, planes8910, off);
               call $step5 (x, y, xstep, ystep, open_sequence, raw);
               end;
          else if z = 4 then do;
               do z = 4 while (z = 4);
                    call $tablet (x, y, z);
                    end;
               call askn ("--> DO YOU REALLY WANT TO QUIT? (yes/no) ",
                    answer);
               if answer = yes then do;
                    raw->font.serial_num = raw->font.serial_num - 1;
                    goto wrapup;
                    end;
```

```
-----------------------------------------------------------------
4    >u>nardy>edge>c5.pl1
-----------------------------------------------------------------

               end;
          else if z = 8 then do;
               do z = 8 while (z = 8);
                    call $tablet (x, y, z);
                    end;
               call ioa ("END CHARACTER");
               call $step5 (x, y, xstep, ystep, close_sequence, raw);
               call grin$linev (0, raw->font.base_line, 511,
                    raw->font.base_line, plane8, on);
               call grin$linev (0, raw->font.x_line, 511,
                    raw->font.x_line, plane8, on);
               call grin$linev (0, raw->font.cap_line, 511,
                    raw->font.cap_line, plane8, on);
               call $vidicon;
               end;
          end;
     do z = 1 while (z = 1);
          call $tablet (xorg, yorg, z);
          end;

     do instruction = look_for_edge while
          ((instruction ~= bad_start) & (instruction ~= open_contour));
          call check_ahead (xorg + 1, yorg, xorg, yorg, instruction);
          xorg = xorg + 1;
          if xorg > 511 then instruction = bad_start;
          end;
     if instruction = open_contour then do;
          call $step5 (xorg, yorg, xstep, ystep, instruction, raw);
          x = xorg;
          y = yorg;
          do while (~((instruction = close_forward) |
                    (instruction = close_fwd_right)));
               call grin$rectv (x, y, x, y, plane10, on);
               call check_ahead  /* straight ahead */
                    (x + xstep, y + ystep, xorg, yorg, instruction);
               if instruction = check_again then do;
                    call check_ahead /* ahead and to the right */
                         (x + xstep + ystep, y + ystep - xstep,
                         xorg, yorg, instruction);
                    call $step5 (x, y, xstep, ystep, instruction, raw);
                    end;
               else call $step5 (x, y, xstep, ystep, instruction, raw);
               if instruction = close_file then goto wrapup;
               end;
          end;
     end;

wrapup:
call $review (raw);
```

```
-------------------------------------------------------------------
5   >u>nardy>edge>c5.pl1
-------------------------------------------------------------------

call ioa ("^rsaving '^a' in directory '>u>type>raw'", font_file);
call hcs$terminate (raw);

end;
```

```
------------------------------------------------------------------
1   >u>nardy>edge>step5.pl1, 03:43:44 Thursday April 28, 1983
------------------------------------------------------------------

step5: procedure (x, y, xstep, ystep, code, raw);

%include raw_file_structure;

declare
     (x, y)                fix(15),
     (xstep, ystep)        fix(15),
     code                  fix(15);

declare
     (ioa, ioskn)          entry  options (variable),
     $summary5             entry (fix, fix, fix, fix, fix, fix, fix);

declare
     counter               fix(15) static,
     pos                   fix(15) static,
     (low_x, high_x)       fix(15) static,
     (low_y, high_y)       fix(15) static,
     (prev_x, prev_y)      fix(15) static,
     prev_close_counter    fix(15) static,
     prev_close_pos        fix(15) static;

declare
     safe_limit            fix(15) init (15000),
     xtemp                 fix(15),
     vector_index          fix(15),
     vector_length         fix(15),
     fixnum                fix(15),
     bit_str               fix(15) defined fixnum;

declare
     open_sequence         fix(15) init (1),
     close_sequence        fix(15) init (2),
     open_contour          fix(15) init (3),
     close_forward         fix(15) init (4),
     step_forward          fix(15) init (5),
     close_fwd_right       fix(15) init (6),
     step_fwd_right        fix(15) init (7),
     rotate_cc             fix(15) init (8),
     close_file            fix(15) init (13);

/*****************************************************************/

if code = open_sequence then do;
     counter = raw->font.chr[raw->font.serial_num].index + 4;
     pos = 1;
     prev_close_counter = raw->font.chr[raw->font.serial_num].index + 6;
     prev_close_pos = 1;
     low_x = 511;
```

```
-----------------------------------------------------------------------
2   >u>nardy>edge>step5.pl1
-----------------------------------------------------------------------

     high_x = 0;
     low_y = 511;
     high_y = 0;
     end;
else if counter => safe_limit then do;
     call ioa ("--> contour vector is full");
     substr (raw->font.vector [counter], 1, 4) = '1010'b;
     code = close_file;
     end;
else if code = open_contour then do;
     xstep = 0;
     ystep = -1;
     fixnum = x;     /* alias bit_str */
     raw->font.vector [counter] = bit_str;
     fixnum = y;     /* alias bit_str */
     raw->font.vector [counter + 1] = bit_str;
     prev_x = x;
     prev_y = y;
     counter = counter + 2;
     end;
else if code = close_forward then do;
     substr (raw->font.vector [counter], pos, 4) = '0000'b;
     prev_close_counter = counter;
     prev_close_pos = pos;
     counter = counter + 1;
     pos = 1;
     x = x + xstep;
     y = y + ystep;
     end;
else if code = close_fwd_right then do;
     substr (raw->font.vector [counter], pos, 4) = '0000'b;
     prev_close_counter = counter;
     prev_close_pos = pos;
     counter = counter + 1;
     pos = 1;
     x = x + xstep;
     y = y + ystep;
     xtemp = ystep;       /* rotate_clockwise */
     ystep = -xstep;      /* rotate_clockwise */
     xstep = xtemp;       /* rotate_clockwise */
     x = x + xstep;
     y = y + ystep;
     end;
else if code = close_sequence then do;
     substr (raw->font.vector [prev_close_counter],
          prev_close_pos, 4) = '1010'b;
     call $summary5 (low_x, high_x, low_y, high_y, counter,
          counter - raw->font.chr[raw->font.serial_num].index,
          raw->font.serial_num);
```

```
-----------------------------------------------------------------
3    >u>nardy>edge>step5.pll
-----------------------------------------------------------------

     fixnum = low_x;      /* alias bit_str */
     raw->font.vector[raw->font.chr[raw->font.serial_num].index] = bit_str;
     fixnum = low_y;      /* alias bit_str */
     raw->font.vector[raw->font.chr[raw->font.serial_num].index + 1] = bit_str;
     fixnum = high_x;     /* alias bit_str */
     raw->font.vector[raw->font.chr[raw->font.serial_num].index + 2] = bit_str;
     fixnum = high_y;     /* alias bit_str */
     raw->font.vector[raw->font.chr[raw->font.serial_num].index + 3] = bit_str;
     call askn ("name of the current character: ",
          raw->font.chr[raw->font.serial_num].name);
     call ioa ("^r");
     raw->font.serial_num = raw->font.serial_num + 1;
     if raw->font.serial_num > 128 then do;
          call ioa
          ("--> character table is now full (128 characters already stored)");
          code = close_file;
          end;
     counter = counter + 1;
     pos = 1;
     raw->font.chr[raw->font.serial_num].index = counter;
     end;
else if code = rotate_cc then do;
     xtemp = -ystep;
     ystep = xstep;
     xstep = xtemp;
     end;
else do;
     if code = step_forward then do;
          x = x + xstep;
          y = y + ystep;
          end;
     if code = step_fwd_right then do;
          x = x + xstep;
          y = y + ystep;
          xtemp = ystep;       /* rotate_clockwise */
          ystep = -xstep;      /* rotate_clockwise */
          xstep = xtemp;       /* rotate_clockwise */
          x = x + xstep;
          y = y + ystep;
          end;

     if x < prev_x then
          substr (raw->font.vector [counter], pos, 2) = '11'b;
     else if x = prev_x then
          substr (raw->font.vector [counter], pos, 2) = '00'b;
     else substr (raw->font.vector [counter], pos, 2) = '01'b;

     if y < prev_y then
          substr (raw->font.vector [counter], pos + 2, 2) = '11'b;
```

```
-----------------------------------------------------------------
4    >u>nardy>edge>step5.pll
-----------------------------------------------------------------

     else if y = prev_y then
          substr (raw->font.vector [counter], pos + 2, 2) = '00'b;
     else substr (raw->font.vector [counter], pos + 2, 2) = '01'b;

     pos = pos + 4;                  /* incr position within string     */
     if pos > 16 then do;            /* if past end of string then...   */
          counter = counter + 1;     /* incr counter to next string     */
          pos = 1;                   /* set position = 1 in next string */
          end;

     prev_x = x;
     prev_y = y;
     if x > high_x then high_x = x;
     if x < low_x  then low_x  = x;
     if y > high_y then high_y = y;
     if y < low_y  then low_y  = y;
     end;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>edge>summary5.pl1, 03:34:13 Thursday April 28, 1983
------------------------------------------------------------------

summary5: procedure (low_x, high_x, low_y, high_y, index, length, serial_num);

declare
     ioa                 entry options (variable),
     grin$rectv          entry (fix, fix, fix, fix, bit(16), fix);

declare
     (low_x, high_x)     fix(15),
     (low_y, high_y)     fix(15),
     index               fix(15),
     length              fix(15),
     serial_num          fix(15),
     ratio               fix(15),
     plane9              bit(16) init ('0200'b4),
     on                  fix(15) init (1),
     (x_extent,y_extent) fix(31),
     xy_pixels           fix(31);

/*****************************************************************/

call grin$rectv (low_x,  low_y,  high_x, low_y,  plane9, on);
call grin$rectv (high_x, low_y,  high_x, high_y, plane9, on);
call grin$rectv (high_x, high_y, low_x,  high_y, plane9, on);
call grin$rectv (low_x,  high_y, low_x,  low_y,  plane9, on);

x_extent  = high_x - low_x + 1;
y_extent  = high_y - low_y + 1;
xy_pixels = x_extent * y_extent;
ratio     = (xy_pixels) / length;

call ioa ("summary:");
call ioa ("    character number         ^5i    (limit 128)", serial_num);
call ioa ("    now up to vector element ^5i    (limit 15000)", index);
call ioa ("    high_x = ^3i,  high_y = ^3i",
     high_x, high_y);
call ioa ("    low_x  = ^3i,  low_y  = ^3i",
     low_x, low_y);
call ioa ("    image size = ^i bytes (1 byte/pixel)", xy_pixels);
call ioa ("    code size  = ^i bytes", length);
call ioa ("    compression ratio image:code =  ^i:1", ratio);

end;
```

```
------------------------------------------------------------------
1   >u>nardy>edge>review.pl1, 03:48:06 Thursday April 28, 1983
------------------------------------------------------------------

review: procedure (raw);

%include raw_file_structure;

declare
     (ioa, ioan, askn)   entry options (variable),
     iocs$getc      entry (char(1)),
     math$sin       entry (float(23)) returns (float(23)),
     math$cos       entry (float(23)) returns (float(23)),
     grin$linev     entry (fix, fix, fix, fix, bit(16), fix),
     grin$rectv     entry (fix, fix, fix, fix, bit(16), fix),
     grin$rect      entry (fix, fix, fix, fix, bit(32));

declare
     code_y              fix(15),  /* current contour code origin     */
     code_x              fix(15),
     bit_x               bit(16) defined code_x,  /* allows access to */
     bit_y               bit(16) defined code_y,  /* bit array as fix */
     (prev_code_x, prev_code_y)
                         fix(15),  /* previous contour code origin    */
     index               fix(15),  /* current element of bit array    */
     pos                 fix(15);  /* position within bit string      */

declare
     (cur_x, cur_y)      fix(15),  /* current integer point           */
     (last_x, last_y)    fix(15);  /* previous integer point          */

declare
     (flx, fly)     float(23),     /* current float point */
     (first_flx, first_fly)
                    float(23),     /* current contour float origin    */
     degrees        float(23),
     radians        float(23),
     repro_scale    float(23),
     1 xstep,
          2 x       float(23),
          2 y       float(23),
     1 ystep,
          2 x       float(23),
          2 y       float(23);

declare
     i              fix(15),
     any_key        char(1),
     planes8910     bit(16) init ('0700'b4), /* erase all        */
     plane8         bit(16) init ('0100'b4), /* blue baseline    */
     plane9         bit(16) init ('0200'b4), /* green box        */
     planes910      bit(16) init ('0600'b4), /* white contour    */
     off            fix(15) init (0),
```

```
------------------------------------------------------------------
2   >u>nardy>edge>review.pl1
------------------------------------------------------------------

     on             fix(15) init (1);

/*****************************************************************/

inner_workings: procedure (i);
     declare
          i         fix(15),
          index     fix(15),
          fixnum    fix(15),
          bit_str   bit(16) defined fixnum,
          (low_x, high_x, low_y, high_y, left_line)
                    fix(15);

index = raw->font.chr[i].index;

bit_str = raw->font.vector [index];
low_x = fixnum;
bit_str = raw->font.vector [index + 1];
low_y = fixnum;
bit_str = raw->font.vector [index + 2];
high_x = fixnum;
bit_str = raw->font.vector [index + 3];
high_y= fixnum;

call grin$linev (low_x,  low_y,  high_x, low_y,  plane9, on);
call grin$linev (high_x, low_y,  high_x, high_y, plane9, on);
call grin$linev (high_x, high_y, low_x,  high_y, plane9, on);
call grin$linev (low_x,  high_y, low_x,  low_y,  plane9, on);
call grin$linev
     (0, raw->font.base_line, 511, raw->font.base_line, plane8, on);
call grin$linev
     (0, raw->font.x_line,    511, raw->font.x_line,    plane8, on);
call grin$linev
     (0, raw->font.cap_line,  511, raw->font.cap_line,  plane8, on);

/* now adjust wrt baseline and left side */
low_y  = low_y  - raw->font.base_line;
high_y = high_y - raw->font.base_line;
high_x = high_x - low_x;
left_line = low_x;
low_x  = 0;
fixnum = low_x;
raw->font.vector [index] = bit_str;
fixnum = low_y;
raw->font.vector [index + 1] = bit_str;
fixnum = high_x;
raw->font.vector [index + 2] = bit_str;
fixnum = high_y;
raw->font.vector [index + 3] = bit_str;
```

```
-------------------------------------------------------------------
3   >u>nardy>edge>review.pl1
-------------------------------------------------------------------


xstep.x = 1;
xstep.y = 0;
ystep.x = 0;
ystep.y = 1;

bit_x = raw->font.vector [index + 4];    /* alias for code_x */
bit_y = raw->font.vector [index + 5];    /* alias for code_y */
/* now refigure starting point wrt baseline and leftline */
code_x = code_x - left_line;
code_y = code_y - raw->font.base_line;
raw->font.vector[index + 4] = bit_x;
raw->font.vector[index + 5] = bit_y;
prev_code_x = code_x;
prev_code_y = code_y;
index       = index + 6;
pos         = 1;

flx         = code_x;
fly         = code_y;
first_flx   = flx;
first_fly   = fly;

cur_x       = code_x;
cur_y       = code_y;
last_x      = code_x;
last_y      = code_y;

do while (substr (raw->font.vector [index], pos, 4) ^= '1010'b);

     if substr (raw->font.vector [index], pos, 4) = '0000'b then do;

          cur_x = first_flx;
          cur_y = first_fly;
          call grin$linev (last_x + left_line, last_y + raw->font.base_line,
               cur_x + left_line, cur_y + raw->font.base_line, planes910, on);
          index = index + 1;
          pos = 1;
          bit_x = raw->font.vector [index];        /* alias for code_x */
          bit_y = raw->font.vector [index + 1];    /* alias for code_y */
          /* refigure starting point wrt baseline and leftline */
          code_x = code_x - left_line;
          code_y = code_y - raw->font.base_line;
          raw->font.vector[index]     = bit_x;
          raw->font.vector[index + 1] = bit_y;

          index = index + 2;

          first_flx = first_flx + ((code_x - prev_code_x) * xstep.x);
```

```
----------------------------------------------------------------
4   >u>nardy>edge>review.pl1
----------------------------------------------------------------

          first_fly = first_fly + ((code_x - prev_code_x) * xstep.y);
          first_flx = first_flx + ((code_y - prev_code_y) * ystep.x);
          first_fly = first_fly + ((code_y - prev_code_y) * ystep.y);

          prev_code_x = code_x;
          prev_code_y = code_y;
          flx = first_flx;
          fly = first_fly;
          cur_x = first_flx;
          cur_y = first_fly;

          end;

     else do;
          if substr (raw->font.vector [index], pos, 2) = '01'b then do;
               flx = flx + xstep.x;
               fly = fly + xstep.y;
               end;
          else if substr (raw->font.vector [index], pos, 2) = '11'b then do;
               flx = flx - xstep.x;
               fly = fly - xstep.y;
               end;
          if substr (raw->font.vector [index], pos + 2, 2) = '01'b then do;
               flx = flx + ystep.x;
               fly = fly + ystep.y;
               end;
          else if substr (raw->font.vector [index], pos+2, 2) = '11'b then do;
               flx = flx - ystep.x;
               fly = fly - ystep.y;
               end;
          cur_x = flx;
          cur_y = fly;
          call grin$linev (last_x + left_line, last_y + raw->font.base_line,
               cur_x + left_line, cur_y + raw->font.base_line, planes910, on);
          pos = pos + 4;
          if pos > 16 then do;
               index = index + 1;
               pos = 1;
               end;
          end;
     last_x = cur_x;
     last_y = cur_y;
     end;

cur_x = first_flx;
cur_y = first_fly;
call grin$linev (last_x + left_line, last_y + raw->font.base_line,
     cur_x + left_line, cur_y + raw->font.base_line, planes910, on);
end; /* end of inner workings */
```

```
------------------------------------------------------------------
5    >u>nardy>edge>review.pl1
------------------------------------------------------------------


/*****************************************************************/

call grin$rectv (0, 0, 511, 511, planes8910, off);
call grin$rect  (0, 0, 511, 511, '00000000'b4);
do i = 1 to raw->font.serial_num;
     call ioa ("^rnow drawing serial_num[^i] = ^a", i, raw->font.chr[i].name);
     call inner_workings (i);
     call ioa ("...to continue, press any key");
     call iocs$getc (any_key);
     call grin$rectv (0, 0, 511, 511, planes8910, off);
     end;

raw->font.cap_line = raw->font.cap_line - raw->font.base_line;
raw->font.x_line = raw->font.x_line - raw->font.base_line;
raw->font.base_line = 0;

end;
```

```
-------------------------------------------------------------------
1   >u>nardy>edge>vidicon.pl1, 03:35:25 Thursday April 28, 1983
-------------------------------------------------------------------

vidicon: procedure;      /*  code taken from vid_in.pl1/gtest.pl1  */

declare
     (ioa, ioan)     entry options (variable),
     iocs$getc       entry (char(1)),
     grin$gwrite     entry ( ,fix);

declare
     spd1            bit(16) init('a002'b4), /* select digitizer card       */
     spd0            bit(16) init('a000'b4), /* initialize all peripherals  */
     lpr_cd          bit(16) init('c000'b4), /*select shift/thresh modes    */
     lpd_cd          bit(16) init('d800'b4), /*select continuous digitizing */
     inbuf [12]      bit(16),
     any_key         char(1),
     (x, y, z)       fix(15),
     break           condition;

on break begin;
     goto exit;
     end;

/* setup */
inbuf [1] = '803f'b4;
inbuf [2] = '1fff'b4;
call grin$gwrite (inbuf, 2);

/* digitize */
inbuf [1] = spd1;   /* select digitizer card           */
inbuf [2] = lpr_cd; /* select no shift, no threshold   */
inbuf [3] = lpd_cd; /* select continuous digitizing    */
call grin$gwrite (inbuf, 3);

call ioan ("^r--> Press spacebar to stop digitizing");
call iocs$getc (any_key);
call ioa ("");

/* clean up */
inbuf [1] = spd0;
call grin$gwrite (inbuf, 1);

exit:;

end;
```

```
------------------------------------------------------------------
1   >u>nardy>edge>alignment.pl1, 03:56:20 Thursday April 28, 1983
------------------------------------------------------------------

alignment: procedure (raw);

%include raw_file_structure;

declare
     (ioa, ioan)     entry options (variable),
     askn            entry options (variable),
     iocs$getc       entry (char(1)),
     $vidicon        entry,
     $tablet         entry (fix, fix, fix),
     grin$rectv      entry (fix, fix, fix, fix, bit(16), fix),
     grin$linev      entry (fix, fix, fix, fix, bit(16), fix);

declare
     choice          char(1),
     (x, y, z)       fix(15),
     prev_y          fix(15),
     off             fix(15) init (0),
     on              fix(15) init (1),
     plane8          bit(16) init ('0100'b4), /* blue lines */
     planes8910      bit(16) init ('0700'b4); /* erase all  */

/*****************************************************************/

get_numbers: procedure;

     call ioa ("umeric values for lines currently are: ");
     call ioa ("    cap height located at ^i", raw->font.cap_line);
     call ioa ("    x height located at   ^i",   raw->font.x_line);
     call ioa ("    base line located at  ^i",  raw->font.base_line);
     call grin$rectv (0, 0, 511, 511, planes8910, off);

     call askn ("^r enter new location of base line (0-511):       ",
          raw->font.base_line);
     call grin$linev
          (0, raw->font.base_line, 511, raw->font.base_line, plane8, on);

     call askn ("   enter new location of cap-height line (0-511): ",
          raw->font.cap_line);
     call grin$linev
          (0, raw->font.cap_line, 511, raw->font.cap_line, plane8, on);

     call askn ("   enter new location of x-height line (0-511):   ",
          raw->font.x_line);
     call grin$linev
          (0, raw->font.x_line, 511, raw->font.x_line, plane8, on);
     end;

/*****************************************************************/
```

```
------------------------------------------------------------------
2   >u>nardy>edge>alignment.pl1
------------------------------------------------------------------


display_lines: procedure;

     call ioa ("isplay lines:");
     call ioa ("    cap height located at ^i", raw->font.cap_line);
     call ioa ("    x height located at   ^i",   raw->font.x_line);
     call ioa ("    base line located at  ^i",  raw->font.base_line);
     call grin$rectv (0, 0, 511, 511, planes8910, off);
     call grin$linev
          (0, raw->font.base_line, 511, raw->font.base_line, plane8, on);
     call grin$linev
          (0, raw->font.x_line, 511, raw->font.x_line, plane8, on);
     call grin$linev
          (0, raw->font.cap_line, 511, raw->font.cap_line, plane8, on);
     end;

/*****************************************************************/

set_cap: procedure;

     call ioa ("ap height via tablet:");
     call ioa ("    press 1 to position cap line");
     call ioa ("    press 2 to move cap line down 1 pixel");
     call ioa ("    press 3 to accept position");
     call ioa ("    press 4 to move cap line up 1 pixel");
     prev_y = raw->font.cap_line;
     call $tablet (z, y, z); /* flush */
     do z = 0 while (z ^= 4);

          do z = 0 while (z < 1);  /* await press */
               call $tablet (x, y, z);
               end;

          if z = 1 then do;
               do z = 1 while (z ^= 0); /* await release */
                    call $tablet (x, raw->font.cap_line, z);
                    call grin$linev (0, prev_y, 511, prev_y, plane8, off);
                    call grin$linev (0, raw->font.cap_line, 511,
                         raw->font.cap_line, plane8, on);
                    prev_y = raw->font.cap_line;
                    end;
               end;

          else if z = 2 then do;
               do z = 2 while (z = 2); /* await release */
                    call $tablet (x, y, z);
                    end;
               raw->font.cap_line = raw->font.cap_line - 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
```

```
-----------------------------------------------------------------
3    >u>nardy>edge>alignment.pl1
-----------------------------------------------------------------

               call grin$linev (0, raw->font.cap_line, 511,
                    raw->font.cap_line, plane8, on);
               prev_y = raw->font.cap_line;
               end;

          else if z = 8 then do;
               do z = 8 while (z = 8); /* await release */
                    call $tablet (x, y, z);
                    end;
               raw->font.cap_line = raw->font.cap_line + 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
               call grin$linev (0, raw->font.cap_line, 511,
                    raw->font.cap_line, plane8, on);
               prev_y = raw->font.cap_line;
               end;
          end;
     call ioa ("--> cap line is located at ^i^r", raw->font.cap_line);
     end;

/*****************************************************************/

set_x: procedure;

     call ioa (" height via tablet:");
     call ioa ("    press 1 to position x line");
     call ioa ("    press 2 to move x line down 1 pixel");
     call ioa ("    press 3 to accept position");
     call ioa ("    press 4 to move x line up 1 pixel");
     prev_y = raw->font.x_line;
     call $tablet (z, y, z); /* flush */
     do z = 0 while (z ^= 4);

          do z = 0 while (z < 1);  /* await press */
               call $tablet (x, y, z);
               end;

          if z = 1 then do;
               do z = 1 while (z ^= 0); /* await release */
                    call $tablet (x, raw->font.x_line, z);
                    call grin$linev (0, prev_y, 511, prev_y, plane8, off);
                    call grin$linev (0, raw->font.x_line, 511,
                         raw->font.x_line, plane8, on);
                    prev_y = raw->font.x_line;
                    end;
               end;

          else if z = 2 then do;
               do z = 2 while (z = 2); /* await release */
                    call $tablet (x, y, z);
```

```
------------------------------------------------------------------
4   >u>nardy>edge>alignment.pl1
------------------------------------------------------------------

                    end;
               raw->font.x_line = raw->font.x_line - 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
               call grin$linev (0, raw->font.x_line, 511,
                    raw->font.x_line, plane8, on);
               prev_y = raw->font.x_line;
               end;

          else if z = 8 then do;
               do z = 8 while (z = 8); /* await release */
                    call $tablet (x, y, z);
                    end;
               raw->font.x_line = raw->font.x_line + 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
               call grin$linev (0, raw->font.x_line, 511,
                    raw->font.x_line, plane8, on);
               prev_y = raw->font.x_line;
               end;
          end;
     call ioa ("--> x line is located at ^i^r", raw->font.x_line);
     end;

/*****************************************************************/

set_base: procedure;

     call ioa ("ase height via tablet:");
     call ioa ("    press 1 to position base line");
     call ioa ("    press 2 to move base line down 1 pixel");
     call ioa ("    press 3 to accept position");
     call ioa ("    press 4 to move base line up 1 pixel");
     prev_y = raw->font.base_line;
     call $tablet (z, y, z); /* flush */
     do z = 0 while (z ^= 4);

          do z = 0 while (z < 1);  /* await press */
               call $tablet (x, y, z);
               end;

          if z = 1 then do;
               do z = 1 while (z ^= 0); /* await release */
                    call $tablet (x, raw->font.base_line, z);
                    call grin$linev (0, prev_y, 511, prev_y, plane8, off);
                    call grin$linev (0, raw->font.base_line, 511,
                         raw->font.base_line, plane8, on);
                    prev_y = raw->font.base_line;
                    end;
               end;
          else if z = 2 then do;
```

```
------------------------------------------------------------------
5   >u>nardy>edge>alignment.pll
------------------------------------------------------------------

               do z = 2 while (z = 2); /* await release */
                    call $tablet (x, y, z);
                    end;
               raw->font.base_line = raw->font.base_line - 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
               call grin$linev (0, raw->font.base_line, 511,
                    raw->font.base_line, plane8, on);
               prev_y = raw->font.base_line;
               end;

          else if z = 8 then do;
               do z = 8 while (z = 8); /* await release */
                    call $tablet (x, y, z);
                    end;
               raw->font.base_line = raw->font.base_line + 1;
               call grin$linev (0, prev_y, 511, prev_y, plane8, off);
               call grin$linev (0, raw->font.base_line, 511,
                    raw->font.base_line, plane8, on);
               prev_y = raw->font.base_line;
               end;
          end;
     call ioa ("--> base line is located at ^i^r", raw->font.base_line);
     end;

/*****************************************************************/

raw->font.cap_line   = 400;
raw->font.x_line     = 300;
raw->font.base_line = 200;

do choice = "" while (choice ~= "q");

     call ioa ("^rOPTIONS:");
     call ioa ("    v  vidicon input
     c  set cap height via tablet
     x  set x height via tablet
     b  set baseline via tablet
     n  set cap-x-base lines numerically via keyboard
     d  display cap-x-base lines on screen, values at terminal
     q  quit");
     call ioan ("^rchoice: ");
     call iocs$getc (choice);

     if choice = "v" then do;
          call ioa ("idicon on");
          call $vidicon;
          call ioa ("vidicon off");
          end;
     else if choice = "c" then call set_cap;
```

```
------------------------------------------------------------------
6    >u>nardy>edge>alignment.pl1
------------------------------------------------------------------

     else if choice = "x" then call set_x;
     else if choice = "b" then call set_base;
     else if choice = "n" then call get_numbers;
     else if choice = "d" then call display_lines;
     end;
call ioa ("uit alignment procedure^r");
end;
```

```
------------------------------------------------------------------
1   >u>nardy>edge>tablet.pl1, 04:02:07 Thursday April 28, 1983
------------------------------------------------------------------

tablet: procedure (x, y, z);

declare
     (x, y, z)       fix(15);

declare
     grin$vis        entry (bit(16)),
     grin$pos        entry (fix(15), fix(15), fix(15)),
     $ss             entry (bit(16), bit(8)),
     $wd             entry (bit(16), bit(8)),
     $rd             entry (bit(16), bit(8));

declare
     /*              i/o ports currently invert data from/to the tablet,      */
     /*              so active state of bits is '0'b                          */
     next_byte       bit(8) aligned init ('01101110'b), /* next_byte set      */
     byte_received   bit(8) aligned init ('10101110'b), /* byte_rec'd set     */
     reset_nbbr      bit(8) aligned init ('11101110'b), /* nb&br reset        */
     all_ones        bit(8) aligned init ('11111111'b),
     all_zeros       bit(8) aligned init ('00000000'b),
     garbage_byte    bit(8),
     first_byte      bit(8),
     check_byte      bit(8),
     raw_data [1:5] bit(8), /* holds data before conversion to fix(15)       */
     tab_addr        bit(16) init ('00aa'b4), /* address of tablet port      */

     i               fix(15),            /* index for raw_data [2:5] loop    */
     wait            fix(15),            /* index for wait loop              */
     duration        fix(15) init (45),  /* duration of wait loop            */
     far_field       fix(15),            /* if > 500 cursor is in far field  */

     xbits           bit(16) defined x,
     ybits           bit(16) defined y,
     zbits           bit(16) defined z,
     sixteen_ones    bit(16) init ('ffff'b4);

/*****************************************************************/

call $ss (tab_addr, check_byte);
if check_byte = '04'b4 then do;
     x = 0; y = 0; zbits = 'fe'b4;
     return;
     end;

do check_byte = all_ones while (substr (check_byte, 1, 1) ^= '0'b);

     do first_byte = all_ones while (substr (first_byte, 1, 1) ^= '0'b);
          call $wd (tab_addr, next_byte);
          far_field = 0;
```

```
2    >u>nardy>edge>tablet.pl1
-------------------------------------------------------------------

          do garbage_byte = all_ones
               while (substr (garbage_byte, 2, 1) ^= '0'b);
               call $rd (tab_addr, garbage_byte);
               far_field = far_field + 1;
               if far_field > 500 then do;
                    x = 0;
                    y = 0;
                    z = -1;
                    return;
                    end;
               end;
          do wait = 1 to duration;
               end;
          call $wd (tab_addr, reset_nbbr);
          call $rd (tab_addr, first_byte);
          call $wd (tab_addr, byte_received);
          do garbage_byte = all_zeros
               while (substr (garbage_byte, 2, 1) ^= '1'b);
               call $rd (tab_addr, garbage_byte);
               end;
          call $wd (tab_addr, reset_nbbr);
          end;

     raw_data [1] = first_byte;
     do i = 2 to 5;
          call $wd (tab_addr, next_byte);
          do garbage_byte = all_ones
               while (substr (garbage_byte, 2, 1) ^= '0'b);
               call $rd (tab_addr, garbage_byte);
               end;
          do wait = 1 to duration;
               end;
          call $wd (tab_addr, reset_nbbr);
          call $rd (tab_addr, raw_data [i]);
          call $wd (tab_addr, byte_received);
          do garbage_byte = all_zeros
               while (substr (garbage_byte, 2, 1) ^= '1'b);
               call $rd (tab_addr, garbage_byte);
               end;
          call $wd (tab_addr, reset_nbbr);
          end;

     call $wd (tab_addr, next_byte);
     far_field = 0;
     do garbage_byte = all_ones
          while (substr (garbage_byte, 2, 1) ^= '0'b);
          call $rd (tab_addr, garbage_byte);
          far_field = far_field + 1;
          if far_field > 500 then do;
```

```
-----------------------------------------------------------------
3    >u>nardy>edge>tablet.pl1
-----------------------------------------------------------------

                x = 0;
                y = 0;
                z = -1;
                return;
                end;
           end;
      do wait = 1 to duration;
           end;
      call $wd (tab_addr, reset_nbbr);
      call $rd (tab_addr, check_byte);
      call $wd (tab_addr, byte_received);
      do garbage_byte = all_zeros
           while (substr (garbage_byte, 2, 1) ^= '1'b);
           call $rd (tab_addr, garbage_byte);
           end;
      call $wd (tab_addr, reset_nbbr);

      end;

xbits = sixteen_ones;
ybits = sixteen_ones;
zbits = sixteen_ones;

substr (zbits, 13, 4) = substr (raw_data [1], 3, 4);
substr (xbits, 11, 6) = substr (raw_data [2], 3, 6);
substr (xbits,  5, 6) = substr (raw_data [3], 3, 6);
substr (ybits, 11, 6) = substr (raw_data [4], 3, 6);
substr (ybits,  5, 6) = substr (raw_data [5], 3, 6);

xbits = ^xbits;
ybits = ^ybits;
zbits = ^zbits;

if      x <   77 then x = 77;
else if x > 2124 then x = 2124;
if      y <  500 then y = 500;
else if y > 1940 then y = 1940;
x = (x -  77) / 4;
y = (y - 500) / 3;

call grin$vis ('0002'b4);
call grin$pos (2, x, y);

end;
```

```
------------------------------------------------------------------------
1   >u>nardy>tablet>tablet.doc, 03:26:29 Thursday April 28, 1983
------------------------------------------------------------------------

documentation for tablet.pl1:

remote control of the tablet

parallel interface connector description:

 1  ground                        13  in 3       rate 4
 2  remote reset                  14  out 3
 3  not used                      15  in 4       mode 1
 4  strappable                    16  out 4
    +5 volts (output)             17  in 5       mode 2
 5  not used                      18  out 5
 6  not used                      19  in 6       byte_received
 7  in Ø       status_valid       2Ø  out 6      byte_available
 8  out Ø                         21  in 7       next_byte
 9  in 1       rate 1             22  out 7
1Ø  out 1                         23  ground
11  in 2       rate 2             24  not used
12  out 2                         25  data strobe


Command byte:

in 7    in 6    in 5    in 4    in 3    in 2    in 1    in Ø
 21      19      17      15      13      11      Ø9      Ø7

next    byte    mode    mode    rate    rate    rate    status
byte    recv     2       1       4       2       1      valid


Data bytes:

                                                        data
out 7   out 6   out 5   cut 4   out 3   out 2   out 1   out Ø   strobe
 22      2Ø      18      16      14      12      1Ø      Ø8      25

        byte
 1      avail    f3      f2      f1      fØ      Ø       Ø       *     [byte 1

 Ø      "        x5      x4      x3      x2      x1      xØ      *     [byte 2

 Ø      "        x11     x1Ø     x9      x8      x7      x6      *     [byte 3

 Ø      "        y5      y4      y3      y2      y1      yØ      *     [byte 4

 Ø      "        y11     y1Ø     y9      y8      y7      y6      *     [byte 5

fØ           z axis value
f3 -> f1     flag
x11 -> xØ    12 bit x coordinate
```

```
------------------------------------------------------------------
2   >u>nardy>tablet>tablet.doc
------------------------------------------------------------------

y11 -> y0     12 bit y coordinate

* data strobe is a negative or positive pulse which emulates byte available,
but which I don't think we use.

a logical '1'b = switch "out" (off) [active]
a logical '0'b = switch "in"  (on)  [inactive]

The status valid bit informs the tablet that a change is desired.

if    status_valid = '1'b  then the bit pad will read the command byte;
else (status_valid = '0'b) the bit pad will search for
     (front panel switch closure) | (remote command w/ active status valid);
therefore, when the host processor is changing the rate or mode setting of
the command byte it should:
     first set status_valid = '0'b;
     then change the desired mode and/or rate setting;
     finally reset status_valid = '1'b;

--> But, I am given to understand that currently the P-E i/o port inverts
data from/to the tablet, so '0' is active.

The desired settings for vlw are:

next_byte:      1  0  0  1  0  0  0  1
inverted:       0  1  1  0  1  1  1  0

byte_received:  0  1  0  1  0  0  0  1
inverted:       1  0  1  0  1  1  1  0


the algorithm for getting 5 bytes of information from the tablet:
(using bit values which are not inverted)

(When we arrive at the top of the loop with a '0'b (inverted '1'b),
we want to fall out of the loop; therefore we initialize at 'f'b
so that we go thru the loop at least once. see tablet.pll */

do while (first_bit_of_check_byte ~= '0'b)

     do while (first_bit_of_first_byte ~= '1'b)
          {synch up to the first byte}

          set next_byte <-- '1'b                              (out)
          initialize far_field_counter <-- 0
          do while (byte_available_bit = '0'b)
               {wait for tablet to present data}
               garbage_byte <-- byte from tablet_port         (in)
               increment far_field_counter +1
```

```
------------------------------------------------------------------
3   >u>nardy>tablet>tablet.doc
------------------------------------------------------------------

               if far_field_counter > 500 then
                    {you've waited long enough. the puck
                    is probably not in proximity of tablet}
                    return
          execute a delay loop to settle data
          reset next_byte <-- '0'b                              (out)
          first_byte <-- byte from tablet_port                  (in)
          set byte_received <-- '1'b                            (out)
               {acknowledge data received}
          do while (byte_available_bit = '1'b)
               {wait for tablet to remove data}
               garbage_byte <-- byte from tablet_port           (in)
          reset byte_received <-- '0'b

     raw_data [1] <-- first_byte
     do the same for raw_data [2] to raw_data [5]

          set next_byte <-- '1'b
          do while (byte_available_bit = '0'b)
               garbage_byte <-- byte from tablet_port
          execute delay loop to settle data
          reset next_byte <-- '0'b
          raw_data [i] <-- byte from tablet_port
          set byte_received <-- '1'b
          do while (byte_available_bit = '1'b)
               garbage_byte <-- byte from tablet_port
          reset byte_received <-- '0'b

     {as a final check, assume the next byte is the first_byte
     of the next sequence of 5 bytes}
     set next_byte <-- '1'b
     initialize far_field_counter <-- 0
     do while (byte_available_bit = '0'b)
          {wait for tablet to present data}
          garbage_byte <-- byte from tablet_port       (in)
          increment far_field_counter +1
          if far_field_counter > 500 then
               {you've waited long enough. the puck
               is probably not in proximity of tablet}
               return
     execute delay loop to settle data
     reset next_byte <-- '0'b
     check_byte <-- byte from tablet_port
     set byte_received <-- '1'b
     do while (byte_available_bit = '1'b)
          garbage_byte <-- byte from tablet_port
     reset byte_received <-- '0'b

*/
```