Editor Design in the Context of Control System Simulation

1986

Leon Fadden
University of Central Florida

Find similar works at: http://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
EDITOR DESIGN IN THE CONTEXT OF CONTROL SYSTEM SIMULATION

BY

LEON FADDEN
B.S.E., University of Central Florida, 1984

THESIS
Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Engineering in the Graduate Studies Program of the College of Engineering University of Central Florida Orlando, Florida

Summer Term
1986
ABSTRACT

Advances in microcomputer display devices and support software during the past decade have made the microcomputer an increasingly popular vehicle for technical education. This is especially apparent in the area of simulation. The pedagog can provide the student of control theory not merely with block diagrams and differential algebra but with high resolution color graphic animations supported by mathematical models whose parameters are easily changed by some editor facility. This mode of control system design and behavior study is both faster and more enjoyable for the student, providing greater continuity, concentration, and learning efficiency.

This paper describes the simulation of a PID two-tank level control system. The system is at most fourth-order and provides a good introduction to system control theory. The model is not unusual, and its nonlinear fourth-order Runge-Kutta solution is straightforward. Simulation itself takes form as (1) a graphical animation in which the user is aware of changing water levels and pipe flows and (2) a numerical multi-column output of system inputs, state variables, and outputs. Both applications are very user-friendly.
A special editor is developed, under which the above applications run. This paper is not a thorough treatment of the control system; a course is required here. Instead, the focus is on the editor and the organization of its Pascal source code. Discussed are a general editor concept and object-oriented code template to which any mathematical driver and associated simulators may be adapted. The editor's source code is designed to be programmer-friendly so that the uninitiated programmer may rapidly assimilate editor structure and continue development.
# TABLE OF CONTENTS

LIST OF TABLES .............................................................. vi
LIST OF FIGURES .............................................................. vii
LIST OF PHOTOS .............................................................. viii

INTRODUCTION ................................................................. 1

CHAPTER 1 - THE CONTROL SYSTEM ........................................... 2
  1. System Overview ....................................................... 2
  2. System Model .......................................................... 7
  3. Model Solution ....................................................... 17
  4. System Responses .................................................... 23

CHAPTER 2 - EDITOR BEHAVIOR .............................................. 32
  1. Introductory Remarks ................................................ 32
  2. Editor Concept ........................................................ 34
  3. The Outer Editor ..................................................... 36
  4. The Inner Editor ..................................................... 49

CHAPTER 3 - EDITOR CODE STRUCTURE .................................... 54
  1. Overview .............................................................. 54
  2. The Inner Editor ..................................................... 69
  3. The Middle Editor ................................................... 77
  4. The Item .............................................................. 85
  5. The Box ............................................................... 86

CHAPTER 4 - THE SIMULATORS ............................................... 94
  1. Overview .............................................................. 94
  2. Animation ............................................................ 95
  3. Numerical Output ................................................... 100

SUMMARY AND CONCLUSIONS ............................................... 107

APPENDIX ........................................................................ 111
  A.OUTPUT - Sample Numerical Output ................................ 111
  A.EDIT - The Outer and Middle Editors ............................ 113
  A.DEFSYS - The Default System Record Constant ................ 166
  A.UTILITIES - General Utilities ..................................... 170
  A.MATH - The Math Driver ............................................ 187
  A.DATA CONTROL - The Inner Editor ............................... 196
  A.ANIMATION - The Graphical Animation .......................... 201
  A.NUMBERS - Columnar Numerical Output .......................... 224
  A.PAGE1 - Display Page 1 ............................................. 248
A.PAGE2 - Display Page 2
A.PAGE3 - Display Page 3
LIST OF TABLES

1. UNITS SELECTION .............................................. 16
2. UNITS USED BY MATH DRIVER ............................... 22
LIST OF FIGURES

1.1 Diagram of Physical System ........................................... 3
1.2 Simple Block Diagram of System ................................. 4
1.3 Exponential Operating Curve for Transmitter, Pump, and Valves ........................................... 9
1.4 Transmitter/Units Adapter Operating Curve ............ 12
1.5 Pump Operating Curve ................................................. 12
1.6 Valve Operating Curve ................................................. 13
1.7 System Response to Step Input Under Proportional Control ........................................... 24
1.8 System Response to Step Input Under Proportional-Integral Control ........................................... 25
1.9 System Response to Step Input Under Proportional-Derivative Control (ALPHA=0.9) ........ 26
1.10 System Response to Step Input Under Proportional-Derivative Control (ALPHA=0.1) ....... 27
1.11 System Response to Step Input Under Proportional-Integral-Derivative Control (ALPHA=0.9) ........................................... 28
1.12 System Response to Step Input Under Proportional-Integral-Derivative Control (ALPHA=0.1) ........................................... 29
1.13 System Level Responses to Step Input Under Proportional Control ........................................... 30
1.14 System Flow Responses to Step Input Under Proportional Control ........................................... 31
2.1 Outer Editor Topology ................................................. 37
LIST OF PHOTOS

1. Ringview at Program Start-up .......................... 38
2. Ringview after Loading of Ring .......................... 39
3. Nodeview Display Page 1 ................................. 44
4. Nodeview Display Page 2 ................................. 45
5. Nodeview Display Page 3 ................................. 46
6. Nodeview Editing ......................................... 50
7. Screen during Animation .................................. 96
8. Numerical Output Showing Design Values ............... 101
9. Numerical Output Showing Current Values .............. 102
10. Scrolling Numerical Output .............................. 103
11. Change of Output Interval ............................... 104
INTRODUCTION

This paper describes the integration of graphical and numerical control system simulations with a high-fluency editor specially designed to enable the user to access, compare, and change with great ease model data in a memory-resident file of model records. The two-tank level control system will be described in detail in Chapter 1. Features supported in the simulation are:

1) Proportional, proportional-integral, proportional-derivative and proportional-integral-derivative controller action.

2) Linear and exponential operating curves for the pump motor, valves, and transmitters.

3) Step, pulse, ramp, and sinusoidal inputs.

4) Variable tank sizes and design operating conditions.

5) Various units for all quantities.

Thus, a given system (model record) may be configured to match a broad range of applications.
CHAPTER 1 - THE CONTROL SYSTEM

Section 1 - System Overview

The two-tank level control system under investigation is shown schematically in Figure 1.1. The process itself consists of two tanks coupled by a pipe running from the bottom of the first tank to the bottom of the second. An output pipe on the second tank may be placed at some adjustable height \( H_3 \). Inputs to the process are flow \( F_1 \) into the first tank, valve opening \( \Theta_1 \) in the connecting pipe, flow \( F_L \) into the second tank, and valve opening \( \Theta_2 \) in the output pipe. Process outputs are the tank liquid levels \( H_1 \) and \( H_2 \), inter-tank flow \( F_2 \), and flow \( F_O \) out of the second tank.

The process is embedded in a negative feedback PID control system. See Figure 1.2. In the context of the control system we are concerned with servo and regulator control of level \( H_2 \) (controlled variable), and we drive the process with \( F_1 \) (manipulated variable). Valve openings \( \Theta_1 \) and \( \Theta_2 \) and load flow \( F_L \) are treated as load variables or disturbance inputs. Note that \( H_1 \) will vary as a side-effect of system efforts to drive or maintain \( H_2 \).
Figure 1.1 Diagram of Physical System.
Figure 1.2 Simple Block Diagram of System.
Our concern with $H_1$ during simulation relates only to potential overflow of tank 1. In fact, tank 1, forming a first-order buffer or lag between $F_1$ and $F_2$, is the major obstacle to effective control, indicating a need here for use and study of second-order PID controllers.

$F_1$ is manipulated by the pump motor, which receives its input voltage from the output of the PID controller, which in turn is driven by an error voltage signal generated by the summer. The summer receives its inputs from two identical level-to-voltage transmitters and outputs the difference between the setpoint level $H_2s$ and the measured level $H_2$. Note that the $H_2s$ transmitter is merely a units adapter (height to voltage), whereas the tank 2 transmitter is an actual electronic liquid level sensing device. A tank 1 transmitter is also indicated but will not be used anywhere in this implementation. (The extra transmitter would, however, find use in a state variable feedback control system.)

A typical servo control scenario is as follows: With the system initially operating at design conditions, a positive step change in $H_2s$ is presented at the input. This leads to an error voltage signal via the units adapter, tank 2 transmitter, and summer. The error voltage is then input to the controller where a pre-specified algorithm (P, PI, PD, or PID) generates a controller output voltage to the
pump motor. The pump motor, by some pre-specified operating curve, produces a flow into tank 1, raising the level in tank 1 and increasing the inter-tank pressure difference, giving rise to F2 and, in turn, to H2. Increased pressure head in tank 2 increases FO, thwarting efforts to raise H2.

A characteristic S-curve response progresses with H1 initially leading H2. As H2 responds, error is reduced. Following the error, F1 is reduced, and H1 response is slowed, stopped, and possibly reversed. H2 meanwhile finishes its response, possibly oscillating several times (generally out of phase) with H1 before reaching a new steady state value at or near the setpoint H2s. All new steady state flows (F1, F2, and FO), voltages (V2s, V2 and VM) and levels (H1 and H2) are greater now than before the step change assuming no changes were made in load variables THETA1, THETA2, and FL. Note that the error voltage EV will again be zero where integral action (PI,PID) is present in the controller. An offset (EV positive) will exist where integral control is not present (P, PD) in the controller.
Section 2 - System Model

The reader should refer to APPENDIX A.MATH while reading the following descriptions of device mathematical structure and model solution. The system model is shown below:

\[ V_{2s} = V_{2}(H_{2s}) \]  
\[ V_{2} = V_{2}(H_{2}) \]  
\[ EV = V_{2s} - V_{2} \]  
\[ \alpha Td \Delta \dot{V}_{m} + \Delta \ddot{V}_{m} = KcTdEV + (Kc/Ti)(\alpha Td + Ti)EV + (Kc/Ti)Ev \]  
\[ \Delta \dot{V}_{m} = KcTdEV + KcEv + (Kc/Ti)Ev \]  
\[ Al H_{1} + F_{2} = F_{1} \]  
\[ A2 H_{2} + FO = F_{2} + FL \]  
\[ F_{1} = F_{1}(V_{m}) \]  
\[ F_{2} = C_{1}(\Theta_{1})\text{sgn}(H_{1} - H_{2})|H_{1} - H_{2}|^{1/2} \]  
\[ FO = C_{2}(\Theta_{2})(H_{2} - H_{3})^{1/2} \quad H_{2} > H_{3} \]

By inspection of components in the forward path (controller, tank 1, and tank 2), the system is at most fourth-order.

Before simulation of a system can begin, the user must supply a system configuration via the editor. There are many pieces of data involved in a complete system description. (Actually, default values are provided for ALL items. See APPENDIX A.DEFSYS, constant record defaultsystem.
The default system, however, generally is not what the user desires.)

The first and most obvious defining data are the physical specifications for tank heights and crosssectional areas: HT1 and A1; HT2 and A2. The height H3 of the output pipe must also be specified. In addition, steady state design operating conditions are specified for H2s, THETA1, THETA2, and FL. These data together with specified device operating curves are used to calculate all other design conditions.

During editing, the user may choose a linear or exponential operating curve for four system devices: transmitter/units adapter, pump motor, valve 1, and valve 2. Wherever exponential operating curves are needed in the simulator, the general equation below is used:

\[ Y = \frac{(e^{bx} - 1)}{(e^b - 1)} \]

See Figure 1.3. Independent variable Y is shown as a function of dependent variable X and shape factor b.

Equations 1.1, 1.2, and 1.3 designate the units adapter, tank 2 transmitter, and summer, respectively. Here, the user explicitly chooses an operating curve for the tank 2 transmitter; the units adapter is made to operate by the same curve. The governing equations for transmitter/units adapter operation are shown below:
Figure 1.3 Exponential Operating Curves for Transmitter, Pump, and Valves.
Vmin

LIN: \[ V = V_{\text{min}} + \left[ \frac{H - H_{\text{min}}}{H_{\text{max}} - H_{\text{min}}} \right] (V_{\text{max}} - V_{\text{min}}) \]

Vmax

H_{\text{min}} < H < H_{\text{max}}

H_{\text{max}} < H

V_{\text{min}}

H > H_{\text{min}}

XPO: \[ V = V_{\text{min}} + \frac{\exp\left[ \frac{b(H - H_{\text{min}})}{H_{\text{max}} - H_{\text{min}}} \right] - 1}{\exp(b) - 1} (V_{\text{max}} - V_{\text{min}}) \]

V_{\text{max}}

H_{\text{min}} < H < H_{\text{max}}

H_{\text{max}} > H

The corresponding graphs are shown in Figure 1.4. Parameters \( H_{\text{2min}}, H_{\text{2max}}, V_{\text{2min}}, V_{\text{2max}}, \) and \( b \) are specified by the user during editing immediately after the above LIN/XPO specification is made. See function TRANS2.

The controller is the most complex component. Equation 1.4a shows the real-world PID algorithm employed. This differs from ideal PID control (1.4b) by introduction of the constant \( \text{ALPHA} \) \((0 < \text{ALPHA} < 1)\) in the case of derivative control. A breakdown of the four controller mode algorithms derived from the real-world PID algorithm is shown below.

P: \[ \Delta V_m = K_c E_v \]

PI: \[ \Delta V_{\dot{m}} = K_c E_v + (K_c/T_i) E_v \]

PD: \[ \alpha T_d \Delta V_{\ddot{m}} + \Delta V_m = K_c T_d \dot{E} v + K_c E_v \]

PID: \[ \alpha T_d \Delta V_{\dddot{m}} + \Delta V_{\ddot{m}} = K_c T_d \ddot{E} v + (K_c/T_i)(\alpha T_d + T_i) \dot{E} v + (K_c/T_i) E_v \]

During editing, the user supplies the controller mode together with one or more of the following parameters: \( \text{ALPHA}, \) controller gain \( K_c, \) integral time \( T_i, \) and derivative
time TD. The control algorithm is now completely specified. See procedure getnumbers.

Equations 1.5 and 1.6 are volumetric flow (not mass) conservation equations for tank 1 and tank 2.

Equation 1.7 designates $F_1$ as some function of the pump motor input voltage. Expressions for pump motor operation are shown below:

$$
\begin{align*}
    \text{LIN: } & F = F_{\text{min}} + \left[ \frac{V-V_{\text{min}}}{V_{\text{max}}-V_{\text{min}}} \right] (F_{\text{max}}-F_{\text{min}}) & V_{\text{min}} \leq V \leq V_{\text{max}} \\
    \text{ } & F_{\text{max}} & V_{\text{max}} > V \\
    \text{XPO: } & F = F_{\text{min}} + \frac{b(V-V_{\text{min}})}{\exp(b)-1} (F_{\text{max}}-F_{\text{min}}) & V_{\text{min}} \leq V \leq V_{\text{max}} \\
    \text{ } & F_{\text{max}} & V_{\text{max}} > V
\end{align*}
$$

The corresponding graphs are shown in Figure 1.5. Again, $V_{\text{min}}$, $V_{\text{max}}$, $F_{\text{min}}$, $F_{\text{max}}$, and $b$ are supplied at this time. See function PUMP.

Equations 1.8 and 1.9 designate $F_2$ and $F_0$ as product functions of valve functions $C_1(\Theta_1)$ and $C_2(\Theta_2)$ and the square root of the appropriate differential pressure head (all pipe flows are turbulent). These valve functions are themselves product functions:

$$
\begin{align*}
    C_1(\Theta_1) &= C V_1 A(\Theta_1) \\
    C_2(\Theta_2) &= C V_2 A(\Theta_2)
\end{align*}
$$
Figure 1.4 Transmitter Operating Curves.

Figure 1.5 Pump Operating Curves.
Figure 1.6 Valve Operating Curves.
where CV1 and CV2 are standard maximum valve flow capacities (flow developed through fully open valve at a 1 psi pressure differential). A1(THETA1) and A2(THETA2) are valve gate open area functions, and THETA is the valve opening in percent. Expressions for the area functions are given below:

LIN: \[ A(\text{THETA}) = \frac{\text{THETA}}{100} \quad 0 \leq \text{THETA} \leq 100 \]

XPO: \[ A(\text{THETA}) = \frac{\exp(b \frac{\text{THETA}}{100}) - 1}{\exp(b)-1} \quad 0 \leq \text{THETA} \leq 100 \]

The corresponding graphs are shown in Figure 1.6. Upper and lower boundaries on THETA naturally lie between 0 and 100 percent, making the shape factor b the only parameter to appear in the above expressions. Thus, parameters CV1 and b are entered here. See function VALVE1 and function VALVE2.

Finally, the user is able to choose from five different profiles for system inputs H2s, THETA1, THETA2, and FL. The core of the H2s input driver, for example, is shown below:
DESN: H2SETPOINT := H2SB;
STEP: begin
  if t<t0h2s then H2SETPOINT := H2SB
  else H2SETPOINT := SPh2s;
end;
PULS: begin
  if t<t0h2s then H2SETPOINT := H2SB else
  if t<tlh2s then H2SETPOINT := SPh2s
  else H2SETPOINT := H2SB
end;
RAMP: begin
  if t<t0h2s then H2SETPOINT := H2SB else
  if t<tlh2s then H2SETPOINT := SP1h2s+(SP2h2s-SP1h2s)/
                  (tlh2s-t0h2s)*t
  else H2SETPOINT := SP2h2s
end;
SINE: begin
  if t<t0h2s then H2SETPOINT := H2SB else
  if t<tlh2s then H2SETPOINT := H2SB+(SPh2s-H2SB)*
                  SIN(Wh2s*t)
  else H2SETPOINT := H2SB
end;

Parameters t0, t1, SP, SP1, SP2, and W are supplied as appropriate when an input profile is specified. The THETA1, THETA2, and FL input drivers follow in an identical manner. See function H2SETPOINT, function VALVE1, function VALVE2, and function LOAD.

The user may also choose from any of the units shown in Table 2.
<table>
<thead>
<tr>
<th>DIMENSION</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>seconds, minutes, hours</td>
</tr>
<tr>
<td>Rate</td>
<td>cycles/sec, cycles/min, cycles/hr</td>
</tr>
<tr>
<td>Length</td>
<td>inches, feet, centimeters, meters</td>
</tr>
<tr>
<td>Area</td>
<td>square in, square ft, square cm, square m</td>
</tr>
<tr>
<td>Flow</td>
<td>cubic ft/sec, cubic ft/min, cubic ft/hr</td>
</tr>
<tr>
<td></td>
<td>liters/sec, liters/min, liters/hr</td>
</tr>
<tr>
<td>Voltage</td>
<td>millivolts, volts</td>
</tr>
<tr>
<td>Valve flow capacity</td>
<td>ft$^{3/2}$/sec, ft$^{3/2}$/min</td>
</tr>
<tr>
<td>Valve opening</td>
<td>percent</td>
</tr>
</tbody>
</table>

Selection of the controller mode, input profiles, operating curves, and units will be discussed in greater detail in chapters 2 and 3.
Section 3 - Model Solution

There are four state equations. Two of these are taken directly from conservation equations 1.5 and 1.6:

\[ \dot{H}_1 = \frac{(F_1 - F_2)}{A_1} \]
\[ \dot{H}_2 = \frac{(F_2 + F_1 - F_0)}{A_2} \]
\[ X_3 = 0 \quad \text{P} \]
\[ EV \quad \text{PI} \]
\[ \frac{(EV - X_3)}{(\alpha TD)} \quad \text{PD} \]
\[ X_4 \quad \text{PID} \]
\[ X_4 = 0 \quad \text{P} \]
\[ 0 \quad \text{PI} \]
\[ 0 \quad \text{PD} \]
\[ \frac{(EV - X_4)}{(\alpha TD)} \quad \text{PID} \]

Algebraic equations:

\[ V_{2s} = V_2(H_2s) \]
\[ V_2 = V_2(H_2) \]
\[ EV = V_{2s} - V_2 \]
\[ \Delta VM = \frac{(KC/TI)X_3 + (KC/(\alpha TI)) [\alpha TD + (1-\alpha)]X_4 + (KC/\alpha)EV}{P} \]
\[ F_1 = F_1(VM) \]
\[ F_2 = C_1(\theta_1) \text{sgn}(H_1 - H_2) |H_1 - H_2|^\frac{1}{2} \]
\[ F_0 = C_2(\theta_2) (H_2 - H_3)^\frac{1}{2} \quad H_2 > H_3 \]
We make the following definitions and summarize.

\[
\begin{bmatrix}
X_1 \\
X_2 \\
X_3 \\
X_4
\end{bmatrix} =
\begin{bmatrix}
H_1 \\
H_2 \\
X_3 \\
X_4
\end{bmatrix} \quad \quad \begin{bmatrix}
Y_1 \\
Y_2 \\
Y_3 \\
Y_4 \\
Y_5 \\
Y_6 \\
Y_7
\end{bmatrix} =
\begin{bmatrix}
V_2s \\
V_2 \\
EV \\
\Delta VM \\
F_1 \\
F_2 \\
FO
\end{bmatrix} \quad \quad \begin{bmatrix}
U_1 \\
U_2 \\
U_3 \\
U_4
\end{bmatrix} =
\begin{bmatrix}
H_2s \\
O_1 \\
O_2 \\
F_1
\end{bmatrix}
\]

In the following equations, the original variable names are retained on the right-hand side for the sake of readability.

State equations: \( \dot{X} = f(X, Y, U) \)
where
\[
\begin{align*}
f_1(X, Y, U) &= \frac{(F_1 - F_2)}{A_1} \\
f_2(X, Y, U) &= \frac{(F_1 + F_L - F_0)}{A_2} \\
f_3(X, Y, U) &= X_4 \\
f_4(X, Y, U) &= \frac{(EV - X_4)}{\alpha TD}
\end{align*}
\]

Algebraic equations: \( G(X, Y, U) = 0 \)
where
\[
\begin{align*}
g_1(X, Y, U) &= V_2(H_2s) - V_2s \\
g_2(X, Y, U) &= V_2(H_2) - V_2 \\
g_3(X, Y, U) &= V_2s - V_2 - EV \\
g_4(X, Y, U) &= (\frac{K_C}{T_I}X_3 + (\frac{K_C}{\alpha T_I})[\alpha TD + (1 - \alpha)T_I]X_4 + (\frac{K_C}{\alpha})EV - VM \\
g_5(X, Y, U) &= F_1(VM + _\text{VM}) - F_1 \\
g_6(X, Y, U) &= C_1(\theta_1)\text{sgn}(H_1 - H_2)|H_1 - H_2|^{1/2} - F_2 \\
g_7(X, Y, U) &= C_2(\theta_2)(H_2 - H_3)^{1/2} - FO
\end{align*}
\]

subject to \( X(0) \) and \( Y(0) \)
where \( x_1(0) = H_1 \)
\( x_2(0) = H_2 \)
\( x_3(0) = 0 \) if present
\( x_4(0) = 0 \) if present
\( y_1(0) = V_{2s} \)
\( y_2(0) = V_2 \)
\( y_3(0) = \bar{E}V = 0 \)
\( y_4(0) = \Delta VM = 0 \)
\( y_5(0) = \bar{F}l \)
\( y_6(0) = \bar{F}2 \)
\( y_7(0) = \bar{F}O \)

The system of equations is solved using a fourth-order Runge-Kutta algorithm with binomial coefficients.

Given \( \dot{X} = f(X,Y,U(t)) \)
and \( O = g(X,Y,U(t)) \)
subject to \( X(0) \) and \( Y(0) \):

\[
x_j^{i+1} = x_j^i + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4)
\]
where
\( K_1 = f_j(U(t_i), X_i, Y_i) \)
\( K_2 = f_j(U(t_i + h/2), X_i + hK_1/2, Y_i) \)
\( K_3 = f_j(U(t_i + h/2), X_i + hK_2/2, Y_i) \)
\( K_4 = f_j(U(t_i + h), X_i + hK_3, Y_i) \)
Specifically,

\[ x_{1i+1} = x_{1i} + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \]
where \( K_1 = \frac{(F_1 - F_2)}{A_1} \)
\( K_2 = \frac{(F_1 - F_2)}{A_1} \)
\( K_3 = \frac{(F_1 - F_2)}{A_1} \)
\( K_4 = \frac{(F_1 - F_2)}{A_1} \)

\[ x_{2i+1} = x_{2i} + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \]
where \( K_1 = \frac{(F_2 + F_1(t_{1} + h/2) - F_0)}{\alpha T_D} \)
\( K_2 = \frac{(F_2 + F_1(t_{1} + h/2) - F_0)}{\alpha T_D} \)
\( K_3 = \frac{(F_2 + F_1(t_{1} + h/2) - F_0)}{\alpha T_D} \)
\( K_4 = \frac{(F_2 + F_1(t_{1} + h) - F_0)}{\alpha T_D} \)

\[ x_{3i+1} = x_{3i} + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \]
where \( K_1 = x_4 \)
\( K_2 = x_4 + hK_1/2 \)
\( K_3 = x_4 + hK_2/2 \)
\( K_4 = x_4 + hK_3 \)

\[ x_{4i+1} = x_{4i} + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4) \]
where \( K_1 = \frac{(E_V - x_4)}{\alpha T_D} \)
\( K_2 = \frac{(E_V - x_4 + hK_1/2)}{\alpha T_D} \)
\( K_3 = \frac{(E_V - x_4 + hK_2/2)}{\alpha T_D} \)
\( K_4 = \frac{(E_V - x_4 + hK_3)}{\alpha T_D} \)
The X vector is calculated iteratively beginning at \( t=0 \), the Y vector being recomputed after each iteration. Procedure getnumbers (APPENDIX A.MATH) clearly identifies the calculations involved, their relative order, and the different algorithms used to calculate \( x_3 \) and \( x_4 \).

For purposes of supporting graphical and numerical output during simulation, an empirically determined step size \( h \) of 1/4 second (0.0041666 minutes) is used. The chosen \( h \) reflects a compromise between overall simulation speed and an acceptable amount of jerkiness in graphical water level movements in configurations where high rates exist.

Moreover, regardless of the units specified during editing, all calculations are performed using the default set of units shown in TABLE 3.
TABLE 3
DEFAULT UNITS
USED BY MATH DRIVER

<table>
<thead>
<tr>
<th>DIMENSION</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>minutes</td>
</tr>
<tr>
<td>Rate</td>
<td>cycles/min</td>
</tr>
<tr>
<td>Length</td>
<td>feet</td>
</tr>
<tr>
<td>Area</td>
<td>square ft</td>
</tr>
<tr>
<td>Flow</td>
<td>cubic ft/min</td>
</tr>
<tr>
<td>Voltage</td>
<td>volts</td>
</tr>
<tr>
<td>Valve flow capacity</td>
<td>(ft^{3/2}/\text{min})</td>
</tr>
<tr>
<td>Valve opening</td>
<td>percent</td>
</tr>
</tbody>
</table>
Section 4 - System Responses

System response to step inputs under P, PI, PD, and PID control will now be examined. Refer to figures 1.7 through 1.14.

Generally, system response under proportional control will become underdamped as $K_c$ increases. The plots of Figure 1.7 show overdamped, critically damped, and underdamped H2 level responses.

Introducing integral control (see Figure 1.8) will increase speed of response but will induce additional ringing, resulting in a longer settling time. Adding derivative action to the controller (see figures 1.11 and 1.12) will help lessen the ringing so induced. Tuning a PID controller is then a matter of adjusting $K_c$, $T_i$, and $T_d$ so as to maximize speed of response and minimize overshoot and oscillation.

Derivative action may be added to a proportional controller to increase speed of response and reduce ringing. This can be seen in Figure 1.9 where a 'real world' ($\alpha = 0.9$) PD controller is used. Notice results are not as encouraging when the more ideal PD controller of Figure 1.10 is used. Pure derivative action, however, can make the system sensitive to noise. Therefore, the preferred configuration is usually the complete PID controller (figures 1.11 and 1.12).
Figure 1.7 System Response to Step Input Under Proportional Control.
Figure 1.8 System Response to Step Input Under Proportional-Integral Control.
Figure 1.9 System Response to Step Input Under Proportional-Derivative Control (ALPHA= 0.9, Nearly 'Real World').
Figure 1.10  System Response to Step Input Under Proportional-Derivative Control (ALPHA= 0.1, Nearly Ideal).
Figure 1.11 System Response to Step Input Under Proportional-Integral-Derivative Control (ALPHA= 0.5).
Figure 1.12  System Response to Step Input Under Proportional-Integral-Derivative Control (ALPHA= 0.1).
Figure 1.13  System Level Responses to Step Input Under Proportional Control.
Figure 1.14 System Flow Responses to Step Input Under Proportional Control.

Flow (cfm)

H2s: 20 to 30 ft

Proportional Control
Kc = 3

Flow (cfm)

Time (min)
CHAPTER 2 - EDITOR BEHAVIOR

Section 1 - Introductory Remarks

In the course of designing any editor, we do well to realize that the editor's behavior—regardless of its organization in space, time, or context (mode)—is determined ultimately by keystrokes and by the state of the editor at the time the keystrokes are made. (We assume no light pens, etc.) Since our computers force us to work with raster-oriented display devices instead of truly photographic image generators by which we can pan and zoom over large two-dimensional vistas of data, we accept the fact that, without such elastic imaging, division of editor function by use of context, mode, window, and such is completely necessary. Until better display devices are built, this fact will remain essentially unchanged.

This state of affairs leads often to the development of editors that use many different keys to do many different things. We want to simplify. As any Pascal programmer knows, building a ramiform hierarchy of menus to organize an application is very compelling (not just obvious) given Pascal's block structuring and strong typing. Such organizations are effective but will not suffice where the
editor's usefulness in an educational environment depends on fluent, variable-scope viewing and changing of data.

An editor usually performs several basic operations on the data being edited, e.g., view, insert, delete, and move. These are obvious in the case of text but apply also to the application at hand where we are dealing with many distinct but interrelated scalar (control mode, input profiles, etc.) and numerical data items.

Intuitively, we strive for a closest-packed (omnisymmetric, polyhedral) functionality and settle for the best we can achieve using real machines and real high-level programming languages. Specifically, we wish to minimize the 'distance' between any two points in the editor, whether moving through the data itself or from some mode to another. This 'distance' should be minimized in the most agreeable way possible, paying attention to matters of practical use. For example, highly frequented editor operations should be more carefully designed for speed and ease than less frequented ones assuming of course that 'speed and ease' in both cases represents a design conflict. In editor-functional space, we quantify and qualify distance in the following ways.
1) The number of keystrokes required to move from one point in the editor data space to any other point.

2) The thought and time required to generate the required sequence of keystrokes.

3) Orientation stability: the confusion and fatigue experienced while dealing with (1) and (2).

Anyone who has used a line editor is keenly aware of the reasons why the foregoing criteria have been established. A line editor relies upon the issuing of not-so-friendly command strings intermixed with data to be inserted, deleted, etc. The editor to be described relies instead upon a carefully structured framework through which the user moves (as above), changing data items as he goes. MOVEMENT is the essence of the editor.

Section 2 - Editor Concept

The editor is organized as a memory-resident ring (circular deque) of nodes (records), each of which is a complete system description containing all information necessary to support simulation. Each node is organized as three pages of data, and only one of these pages may be viewed at a time. Within each page there are several boxes, each having an appropriate header and containing several related data items. The data items consist of a name (the variable identifier), the data itself (scalar or real), and units, where appropriate, arranged horizontally.
Movement through all of these data objects (ring, node, page, box, and item) is accomplished by using the following keys: er, esc, home, end, pgup, pgdn, and the four arrow keys. Use of these keys will be described in detail as we progress. The editing process consists of two major phases:

Phase 1: Beginning with the loading of a disk file and the formation of the ring, MOVE to the target system node, page, box, and finally to the desired data item in the box.

Phase 2: With the cursor now positioned to the immediate right of the data, make one final move to the left into the data field or right into the units field (if present) and CHANGE the data as desired.

Central to the editor concept is the notion of DATA PRESENCE. This means that, based on spatial and motional relationships between visible data formats, the user at all times has a clear understanding of where he is in the global data structure (the ring), where he can go next (and by which keys), and what he can do when he gets there. Stated differently, 'what you see is what you get' at all times. This principle and the above topological characterizations lie at the foundation of user friendliness.

For purposes of describing behavior—-and later code structure—-we associate the term outer editor with the MOVE phase and the terms middle editor and inner editor with the CHANGE phase. Some vocabulary are needed before proceeding with discussion of the MOVE phase:
Ringview: a way of viewing the contents of the ring in abbreviated form such that many system nodes (as many as 18) occupy the screen at once. Gross editing—insertion, deletion, and masking of entire systems—is possible in this view.

Nodeview: a way of viewing the contents of the ring in detail such that only one of the node's three pages occupies the screen at a given time. Editing of the page's boxes' data items is possible in this view.

Section 3 - The Outer Editor

Refer to Figure 2.1 for a graphical summary of outer editor topology. A typical session will now be described.

After compiling the Turbo Pascal editor source code and running the program, the screen shown in Photo 1 will appear. There are three adjoining windows: a small one at the top containing the default file name; another small one at the bottom containing a menu; and a large one in the middle that will contain the ring. (Actually, only the middle one is a true window.) Pressing CR indicates acceptance of the default file; the file is loaded and the corresponding ring appears in the ring window. See Photo 2. This is Ringview. Each line in the ring window represents a system and shows three pieces of information about the system: its three-digit number, its 50 character descriptive name, and the date at which it was last edited. Note that one line—the selected node—is yellow, while the remaining non-active nodes are light gray. The one exception
(a) Outer Editor Topology of Ringview.

(b) Outer Editor Topology of Nodeview.

Figure 2.1 Outer Editor Topology.
Photo 1. Ringview at Program Start-up.
Photo 2. Ringview after Loading of Ring.
is the header node, which is cyan to distinguish it as a non-enterable, non-system node--it is merely a reference point for the ring's beginning and end.

The first line of the menu displays (highlighted) options Work file, Save, and Renumber. Work file moves the cursor from the ring window to the top box, allows the user to specify a file name, and upon or loads the file. If the current ring has not been saved (and has been altered since its last loading), the message '<filename> not saved' will appear briefly in the top box, forcing the user to save the current ring before loading another. Too much data is at stake not to enforce this; accidents cannot be tolerated.

Save, like Work file, first moves the cursor from the ring window to the filename and awaits a change of filename and/or or, then writes the ring to disk. This opportunity to change the filename before saving makes it easy to create duplicate files or to break up a single large ring into several smaller ones using the mask feature of the outer editor (to be described).

A two-file first-in-first-out (FIFO) storage protocol is employed. The suffix on the current file (last) is '.SIM'. The suffix on the backup file (first) is '.BAC'. If a name is typed without a suffix, '.SIM' is assumed. Should the suffix be specified, these are the only allowed ones. If the filename is changed before saving, the user
will be prompted 'Overwrite old <filename> ? (Y/N)' if the specified file already exists on disk.

Renumber renumbers the ring starting from 001 (the header node is 000). This should be done seldom if the user wishes to preserve the ring's chronology. Node numbers are issued sequentially from 001 to 999 at each node insertion (creation). After much editing, gaps in the flow of node numbers will serve to remind the user of systems that have come and gone through the growth of the ring. Once node number 999 has been reached (which may but probably will not indicate a ring of 999 system nodes), the ring cannot be expanded further and must then be renumbered if further expansion is required. The prompt 'Renumber? (Y/N)' is issued in the top box to confirm the user's intention to renumber.

The key-primitive menu is shown in yellow below the key-word menu. Pressing ins causes a new system node to be created and inserted just below the siteline. This system is a default (like the filename) and may be edited and simulated immediately if desired. The node number will be one greater than the largest already in the ring; the default name is the number until a suitable one is entered during Nodeview editing; and the date is the current date.

Similarly, deletion of an entire system is accomplished by pressing del. This is not actually a deletion but a mask
that may be toggled (yellow=active, red=masked). Thus, deletions go into effect only when the ring is saved, at which time the user is prompted 'Complete deletions? (Y/N)'.

The up arrow and down arrow move the yellow siteline up and down, changing the active node. The ring scrolls when the siteline is pushed past the upper and lower boundaries of the window.

Keys pgup and pgdn move the siteline a 'page' of nodes at a time. In the manner of a text editor, the bottom line of the ring window becomes the top line in the case of pgdn, for example, and the physical position of the siteline on the screen is unchange. The ring appears to rotate under the window. To circumnavigate a 1000-node ring, roughly 56 strokes are required--about 22 seconds of continuous key depression on an IBM AT personal computer. Keys spc and cr in Ringview relate only to editing of the filename. The editing or CHANGE protocol is identical for all items in the editor, including the filename. This protocol will be discussed in the context of Nodeview editing.

The user, having come to rest on the system node he wishes examine/edit/simulate, presses cr to zoom on the node, and the screen shown in Photo 3 appears. This is Nodeview. The three-window frame of Ringview is unchanged, but the contents are different. The top window now contains
the siteline (node name), the bottom window contains a new menu, and the middle window contains three boxes of data: Controls, Physical specs, and Design conditions. This is page 1 (another default), as indicated by the small added box in the upper right-hand corner of the page window.

The cursor is now positioned to the right of the node name in the top window (see Photo 3). Before actually editing data items, there are three types of movements that the user can make in Nodeview.

1) **Ring traversal.** Pressing the up and down arrows moves our position in the ring up or down one node, as in Ringview. Two things have changed in Nodeview ring traversal. The first is the detail with which the user can view the contents of the nodes. The same page view (and cursor position) is maintained with each node-to-node movement, this-node data in all boxes being quickly erased and replaced by corresponding next-node data. Since the template for a given page is the same for all system nodes, the user is provided with a convenient and topologically lucid way of comparing the data in many systems on a given page level.

The second change is that system nodes masked in Ringview will not be visible during Nodeview ring traversal, i.e., they will be skipped. Thus, the ring may be short-circuited or pinched-off in sections that are not of current
interest. For example, assume a 100 node ring. The user may move from a given page view of node 1 to the same page view of node 50 with a single keystroke if nodes 2 to 49 and nodes 51 to 100 have been masked. In this manner, the effective distance between nodes—not their relative order—may be changed.

2) Node traversal. Pressing pgup or pgdn will move us through the three display pages associated with each system node. These pages are shown in photos 3, 4, and 5. Pressing Pgdn shows page 1, page 2, page 3, page 1, etc. Pressing pgup shows page 1, page 3, page 2, page 1, etc. Only the middle window is altered with each keystroke: this-page boxes and their contents are quickly erased and replaced by next-page boxes and their contents. This process is called node traversal. Note that the operations assigned to pgup and pgdn in Nodeview are completely different from those in Ringview, as dictated by context.

The user will notice a circular topology at all levels of editor organization. Using a 'cylindrical planet' metaphor, we imagine Ringview traversal as a satellite view of the ring (our planet); the view is comprehensive but coarse. Nodeview traversal, on the other hand, may be likened to low-level flying in which little can be seen at once (a given page of a single node) but data items (surface details) may be quickly changed. The one-dimensional
Ringview ring of Figure 2.1a and the higher resolution two-dimensional Nodeview toriod of Figure 2.1b are now understandable.

3) **Page traversal.** Returning to our initial position after zooming (at the node name atop page 1), we can discuss movement through a given page. Pressing **cr** repeatedly will move the cursor along a circuit beginning with the node name (another default) in the node header through the data items in the first box of the page (moving left to right and down, box to box), then through those of the second box, and so forth until there are no more page boxes, at which time it returns to the node name. After each movement, the cursor comes to rest at a position that is two spaces to the right of the last position in the data field, i.e., there is one space between the end of a full data field and the cursor. In addition to **cr**, keys **home** and **end** can be used: **end** moves the cursor in the same direction as **cr**; **home** moves it in the opposite direction. Pressing the **ctrl-cr** moves the cursor to the first item in the next box to provide for accelerated page traversal.
Section 4 - The Inner Editor

Having arrived at the desired system node's page's box's data item, the user enters the data field or units field (if present) by pressing spc (or left arrow) or Units (or right arrow), respectively. This movement and that of selecting an application (Design or Simulate) is accomplished by the middle editor (discussed in Chapter 3).

The selected data item, formerly yellow, will now turn light gray with the cursor to the immediate right (see Photo 6). The inner editor is now active and will remain active until cr is pressed, at which time the data turns yellow again and the cursor moves back to the center point between the two fields.

While the data is light gray, the user may pick at it in the usual way using keys available in any full screen editor: rub, left arrow, right arrow, and del. The space bar, though, provides additional convenience. Pressing spc when there is at least one character of data in the field erases (bulk erases) the data field completely. At this point, the old data may be retrieved by pressing spc again (toggle style). New data may be typed into a cleared field, erased, and the old data retrieved with two successive strokes of spc. Generally, the old data may be retrieved any time before cr is pressed. Typing will always insert (not overwrite) characters. The user will be warned with a
Photo 6. Nodeview Editing.
beep when he attempts to overfill the field. Upon cr, all strings (names) and scalars are left justified, and all real numbers are right justified.

Associated with each variable in the editor are range parameters (specified in the source code) that determine whether a newly entered value is correct. The following represent the editor's original configuration but may be changed as needed:

**strings:** [!...~]
**control mode:** [P,PI,PD,PID]
**operating curve:** [LIN,XPO]
**input profile:** [DESN,STEP,PULS,RAMP,SINE]
**real numbers:** min = 0.000 to max = 999.990

**units:**

- **Time:** sec, min, hrs
- **Rate:** cps, cpm, cph
- **Length:** in, ft, cm, m
- **Area:** sqin, sqft, sqcm, sqm
- **Flow:** cfs, cfm, cfh, lps, lpm
- **Voltage:** mv, v
- **Valve flow capacity:** ft^(3/2)/sec, ft^(3/2)/min
- **Valve opening:** %

Error control is preventative. If an entered value does not conform to the above values/boundaries, the user will hear a beep. No error message is issued. Instead, the user presses the up and down arrow keys (of course) to obtain ring-stored information on correct values/ranges for
the variable whose data field he now occupies. Correction information is thus provided locally (to save display space and preserve orientation) and immediately using the most obvious keys. In the case of the four scalar sets shown above, the values are simply displayed in the data field one at a time in sequence in the order shown. As an added convenience, the user may enter the editor-supplied value as is. Again, we are only choosing data structures (this time very simple ones) from a ring of many. In the context of a single data item, a micro-cylindrical-planet or thumbwheel metaphor is applicable.

In the case of real numbers, a minimum and a maximum value are displayed in the data field. These may also be entered directly if desired. And in the case of strings (file name and node name), the entire field (12 and 50) is filled with contiguous groups of ordered characters within the limits of the specified character range. That is, each stroke of the up or down arrow key shifts <field> characters out and the next <field> characters in, moving in the indicated direction. These nonsense strings may also be entered as names.

To save time, an on-line units conversion utility is built into the editor. As the user cycles through the units associated with a given variable, the corresponding number
to the left is changing so as to preserve the original physical quantity (number and units taken together).

This completes description of inner and outer editor behavior. Summarizing, movement, photographic metaphor (variable scope), and uniform, recursive application of circular topology to data structure figure most strongly in the design and organization of the editor.
CHAPTER 3 - EDITOR CODE STRUCTURE

Section 1 - Overview

A cursory look through the APPENDIX, especially A.PAGE1, A.PAGE2, and A.PAGE3, will reveal a decidedly object-oriented code structure and concomitant message protocol. Indeed, much of the editor code structure is solid (crystalline) in nature: a few universal code patterns are developed and applied uniformly throughout. The most prominent form observed is the case structure (this programmer's unit cell). It is used profusely throughout the editor, and wherever it appears, one finds that case selectors consist only of constant key identifiers, the aforemention scalar message identifiers, or integers.

Such infatuation with a single Pascal structure is amply justified. The case structure with its selectors accounts explicitly for all actions that might take place within its boundaries, without need of the more tedious if-then-else logical mediation. Selectors applied to but not appearing in the case structure are simply not acknowledged and so cause no complications. Furthermore, the typographical formats achievable using the case structure are extremely blockish, predictable, and readable. Causing
the uninitiated programmer distress is no way to ensure longevity of the source code and the organizational concepts it harbors—no matter how novel.

Another gross feature of the source code is that procedure names and message selectors are placed in simple predicate-object form, consistently designating simple actions related directly to external behavior (putbox, put_box; rubdata, rub_data) not behavior implementation strategies. Also, the names of the the major high-level procedures serve to fortify the object-oriented organizational concept. Generally, to the extent permitted by Pascal (a procedural language), we strive for an isomorphism of behavior and code structure, bearing in mind that, ideally, understanding of program source code should follow directly from a detailed knowledge of external behavior. Again, designs for our behavior-generating machines and the high-level languages that drive them do not reflect omniscience and so preclude this ideality. A hierarchial skeleton of the editor program is shown below:
program twotanks (input, output);

type
constant
var

{general utilities}

procedure data_control (olddata : datastring;
                        var newdata: datastring;
                        xref, line : integer;
                        field : integer;
                        var control: char	en );

begin
  1. Takes data and control key
     information from the keyboard.
  2. Performs simple error checking.
end;

procedure thename (var name : datastring;
                  xref, line : integer;
                  field : integer;
                  nameset : charset;
                  var control: char	en );

begin
  1. Nodeview ring, node, page, and item
     movements initiated here.
  2. Gets names from the keyboard via data_control.
  3. Performs item-specific error checking.
  4. Displays allowable characters.
  5. Permits exit to a menu choice.
end;
procedure thesymbol (var symbol : scalar;
xref, line : integer;
field : integer;
symbolset : scalarset;
var control : char);

begin
  1. Nodeview ring, node, page, and item movements initiated here.
  2. Gets scalars (control mode, input profiles, and opcurves) from the keyboard via data_control.
  3. Performs item-specific error checking.
  4. Displays allowable scalars.
  5. Permits exit to a menu choice.
end;

procedure thequantity (var number : real;
var units : scalar;
xref, line : integer;
field : integer;
min, max : real;
unitset : scalarset;
enforce : boolean;
var control : char);

begin
  1. Nodeview ring, node, page, and item movements initiated here.
  2. Gets number and units from the keyboard via data_control
  3. Performs item-specific error checking.
  4. Displays allowable units and the minimum and maximum number based on these units.
  5. Permits exit to a menu choice.
end;
procedure thisfile;

procedure thisring;

procedure thisnode;

procedure thispage;

procedure page1 (var box, item: integer);

procedure box1;

procedure item1 (dowhat: message);
begin
  with worknode^.system do
  case dowhat of
    putname: align ('item 1=', xref, line);
    putdata: putname ( );
    getdata: thename ( );
  end
end;

procedure putbox;
begin
end;

procedure rubbox;
begin
end;

procedure putdata;
begin
end;

procedure rubdata;
begin
end;
procedure getdata;

    procedure getITEM0;

    begin
        getITEM0;
    end;

begin {getdata}
    case control of
        cr : item:= 0;
        home: item:= 4;
        endk: item:= 0;
        ccr : item:= 0;

    end;
end;

begin {box1}
    case dow what of
        put_box: putbox;
        rub_box: rubbox;
        put_data: putdata;
        rub_data: rubdata;
        get_data: getdata;

    end
end;
begin {pagel}
box1 (put_box);
box2 (put_box);
box3 (put_box);
box4 (put_box);
repeat
box1 (put_data);
box2 (put_data);
box3 (put_data);
box4 (put_data);
repeat
  case box of
    1: box1 (get_data);
    2: box2 (get_data);
    3: box3 (get_data);
    4: box4 (get_data);
  end;
  case control of
    cr : box := box mod 4 + 1;
    home: if box = 1 then box := 4 else box := box - 1;
    endk: box := box mod 4 + 1;
    ccr : box := box mod 4 + 1;
  end;
until (control in [uarr, darr, pgup, pgdn, esc])
or (control in menuset);
if (control in [uarr, darr])
then case control of
  uarr: nodeup;
  darr: nodedown;
end;
until (control in [pgup, pgdn, esc])
or (control in menuset);
end; {pagel}
begin {thispage}
repeat
  case page of
    1: page1 (box[1],item[1]);
    2: page2 (box[2],item[2]);
    3: page3 (box[3],item[3]);
  end;
  case control of
    pgup: if page=1 then page:=3 else page:= page-1;
    pgdn: if page=3 then page:=1 else page:= page+1;
  end;
  if control in menuset then begin
    pagenumerator (rub_box);
    pagenumerator (rub_item);
    menufooter (rub_item);
    case control of
      'D': Design;
      'S': simulate;
    end
  end;
until (control= esc);
nodeheader (rub_item);
pagenumber (rub_box);
pagenumber (rub_item);
menufooter (rub_item);
end;

begin {thisnode}
  ringheader (rub_item);
  menufooter (rub_item);
  rubring;
  thispage;
  putring;
  menufooter (put_item);
  menufooter (put_item);
end;
begin {thisring}
putring;
repeat
  ringwindow;
  data_control (d,d,winxref,siteline,0,control);
  if scancode
  then case control of
    ins : insertnode;
    del : deletenode;
    uarr: nodeup;
    darr: nodedown;
    pgup: pageup;
    pgdn: pagedown;
  end
  else case control of
    cr : thisnode;
    'S': savering;
  end;
  until (control in ['W','R']) and not scancode;
end;

begin {thisfile}
control:= 'W';
repeat
  case control of
    'W' : makering;
    'R' : renumber;
  end;
  thisring;
  until (control in [esc]);
end;

begin {main}
  thisfile
end.
This rather full skeleton gives the reader a better-than-rudimentary understanding of code mechanism, in addition to a general notion of global object-oriented procedural structure. The complete editor programs in the APPENDIX, A.EDIT, A.PAGE1, A.PAGE2, and A.PAGE3, should be inspected if a detailed working knowledge of the editor is sought. After study of the skeleton, the reader should be aware of a very systematic and predictable use of: message passing to divide labor by data object instead of by function, the latter being the prevalent strategy in a procedural language; key identifiers, message identifiers, and integers as selectors in case-structure selection and sequencing of procedures; minimal if-then-else logic and repeat loop to control the these organizational devices. Relatively little of the skeleton differs from what appears in the APPENDIX. There is a tremendous reduction in bulk, however, due to the omission of procedures (some shown degenerate) whose operations are obvious or whose presence might detract from comprehension of the general flow.

In particular, only a single generic page (pagel), box (box1), and item (item1) are represented. There are, of course, three different pages and within each three or four boxes containing collectively one to two dozen data items, each taking form as an individual procedure. The complete page procedures are thus very large and well-ordered but not
complex. Other circumstantial code in many of the procedure bodies has been removed to generate an agreeable skeleton. The following sections will treat the individual lower-level procedural object forms in greater detail. We now elucidate code flow in a comprehensive manner.

Beginning in the main body, we enter directly procedure `thisfile` containing procedures (not shown) related to disk input/output and ring assembly, FIFO storage protocol, renumbering, and so forth. The default operation upon entering `thisfile` is to load a file, after which we display the ring (putring) upon entering `thisring`. (Generally, also, we may renumber the ring.)

`Thisring` is responsible for manipulations of the siteline in Ringview traversal and for the corresponding changes of page box data during Nodeview traversal. Insertion and deletion (masking) of nodes is performed here, too. `Saver`ing is accessed from `thisring`, not from `thisfile`, only because the ring is not redisplayed after saving but must be after loading a new file. Finally, `or` steers execution into `thisnode` (zooming).

`Thisnode` rubs (erases) the Ringview header (file name), footer (menu), and the ring itself, setting up the boxes of the appropriate page (page 1 if zooming for the first time) upon entering `thispage`. Before continuing with `thispage`, notice that we may immediately exit Nodeview with `esc` and
return to Ringview by performing the reverse of the above operations: putting instead of rubbing.

To understand how this transfer of control up and down the object hierarchy works, it is enough to bear in mind that \texttt{data\_control} at all times provides the editor with the last entered control key, whether the key be associated with a data entry and movement (\texttt{cr}) or a movement only (\texttt{uarr}, \texttt{darr}, \texttt{pgup}, \texttt{pgdn}, \texttt{ccr}, \texttt{cr}, \texttt{home}, \texttt{end}). This character is contained in the global variable \texttt{control}.

This page immediately places us in the proper page (say \texttt{pagel}). Before continuing with \texttt{pagel}, notice that exit from \texttt{pagel} will occur under three different conditions: 1) the user presses \texttt{esc} to return to Ringview; 2) the user presses \texttt{pgup} or \texttt{pgdn} to move to the next page in the node; 3) the user presses a menu key (Design or Simulate). The first block checks for a change of page and acts accordingly. The second block checks for a menu key. If a 'D' or an 'S' is detected, the page number and menu box are rubbed in preparation for an application and new menu. Prior exit from \texttt{pagel} erased the boxes in the page window (ring window). Note that the node header remains for purposes of identification and orientation. Finally \texttt{esc} will terminate the repeat loop, after which the node header, page number box, and menu footer are all rubbed in preparation for Ringview.
Page1 provides for traversal of all the items in its three boxes: box2, box3, and box4. Another box—the node header—is included in the page traversal circuit, regardless of the page. This is where the cursor rests initially upon entering the page. (For example, in page1, the node header would correspond to box1, the Controls box to box2, the Physical specs box to box3, and the Design conditions box to box4.) Setup of boxes and box data is followed by an inner repeat loop in which we poll all of the items on a box basis, each box taking care of its own items in the polling process. When it is time to move out of one box and into the next (in either direction), the following case structure increments or decrements box depending on the last entered control key. The inner loop terminates when any phase 1 movement is attempted (recall that phase 2 movements are to the left or right only into and out of the item's data fields). At this point a check is made to determine whether a movement up or down to an adjacent node is called for, whereupon nodeup or nodedown appropriately reassigns global pointer variable worknode^, which at all times points to the active node. The outer repeat structure then iterates on all put_data operations, placing new data in all the boxes, and once again re-enters the inner get_data polling loop.
Box1, as the body shows, is straight forward—and predictable—so based on the foregoing descriptions. We focus on getdata, for it makes use of the middle and inner editors (to be discussed) and, so, is the most complex. Getdata under box1 (or any box) permits traversal over the items of the box in the top-to-bottom or bottom-to-top direction until an attempt is made to move pass the first or last item of the box, at which time the aforesaid box-to-box transition occurs back in page1.

The box1 shown is the more general of two types that appear in the editor, i.e., the first item (a scalar) determines which of the following items will be displayed (if an item is not visible, it cannot be edited). Exactly which items are selected and their placement in the box will be discussed in greater detail later.

Item1 makes reference to the current worknode, and then, depending on the received message, does one of three things: 1) writes (align) the variable name with an equal sign justified on the proper line in the box; 2) writes (putdata) the variable data to the right of the name; 3) retrieves (thesymbol/thesymbol/thequantity and data_control) data and control information from the keyboard. Again, (1) and (2) are straightforward and a closer look at the get_data operation (3) is necessary. Although not shown in the skeleton's item1, every item procedure contains a set of
constants determining its field and what data values constitute a correct entry. Each item procedure is thus very self-contained.

The reader will notice that the procedure attached to the get_data selector is called thename not getname, as might be expected. The reason for this is that procedures thename, thesymbol, and thequantity form a sophisticated control junction in which getting data and control information from the keyboard is only one of five operations performed. The operations performed are identical in all three except for the type of data involved. It is at this procedural level that movements down into the inner editor or up into the outer editor (item, box, page, and ring) are initiated and controlled. Error checking and online, immediately enterable, correction information is provided, and ascent and exit to a selected menu application is also made possible. Thus, these three procedures are collectively called the middle editor.

Finally, at the lowest level of editor organization (the inner editor), procedure data_control is operating to take entered data and control information in accordance with the standard data entry protocol discussed in CHAPTER 2 and pass it back up through the higher object forms.

This completes comprehensive, top-down discussion of code flow and organization. The next sections begin a
bottom-up, detailed discussion of lower editor levels, beginning with the inner editor.

Section 2 - The Inner Editor

Procedure `data_control` is the most important procedure in the entire editor and is identically the `inner` editor. All editor input/output is channeled through this single procedure. It was developed in an effort to save time and effort in programming, for, as any programmer knows, fancy input/output operations often consume most of the programmer's energy and constitute a large portion of the final source code.

Development of local input/output processing and error checking repeatedly produces essentially the same operations and code forms, providing incentive for centralization of such operations by use of a simple data entry protocol applicable to all data types in all situations.

The problem of dealing with data types string (name), scalar (symbol), and real and scalar (number and units or quantity) is solved by converting all variables to their equivalent string representations before entering `data_control`. Procedure `data_control` is shown completely below:
procedure data_control (olddata : datastring;
  var newdata: datastring;
  xref,line : integer;
  field : integer;
  var control: char );

const ctrlset: charset= [cr,spc,bksp];
  scanset: charset= [del,larr,rarr];
  dataset: charset= ['!'..'~'];

var ch: char;
  xbeg,ybeg: integer;
  xend,yend: integer;
  exit: boolean;

procedure getch;
begin
  read (kbd,ch);
  if (ch= esc) and keypressed
    then begin
      read (kbd,ch)
      scancode:= true;
    end
  else scancode:= false;
end;

procedure backspace;
begin
  if (wherex > xbeg)
    then gotoxy (wherex-1,wherey);
  exit:= false;
end;

procedure forespace;
begin
  if (wherex < xbeg+field)
    then gotoxy (wherex+1,wherey);
  exit:= false;
end;
procedure rub;

var delpos: integer;
    newpos: integer;

begin
    if (wherex > xbeg)
    then begin
        newpos:= wherex-1;
        delpos:= wherex-xbeg;
        delete (newdata,delpos,1);
        gotoxy (newpos,wherey);
        write (copy (newdata,delpos,80));
        write (spc);
        gotoxy (newpos,wherey);
        exit:= false
    end;
end;

procedure deletech;

var delpos: integer;
    newpos: integer;

begin
    newpos:= wherex;
    delpos:= 1+(wherex-xbeg);
    delete (newdata,delpos,1);
    gotoxy (newpos,wherey);
    write (copy (newdata,delpos,80));
    write (spc);
    gotoxy (newpos,wherey);
    exit:= false;
end;

function despace (data: datastring): datastring;

var n: integer;
    puredata: datastring;

begin
    puredata:= "";
    for n:= 1 to length (data)
    do if data[n]<spc
        then puredata:= puredata+data[n];
    despace:= puredata;
end;
procedure old;
begin
  gotoxy (xbeg,ybeg);
  if (newdata= '')
    then begin
      newdata:= olddata;
      write (olddata);
    end;
  if (newdata<>olddata)
    then mustsave:= true;
  control:= out;
  exit:= true;
end;

procedure new;
begin
  gotoxy (xbeg,ybeg);
  if (despace (newdata)= '')
    then begin
      newdata:= olddata;
      write (olddata);
    end
  else begin
    write (spc:length (newdata));
    newdata:= '';
    gotoxy (xbeg,ybeg);
  end;
  exit:= false;
end;

procedure move;
begin
  if scancode
    then begin
      case ch of
        del : deletech;
        larr: backspace;
        rarr: foreshape;
        end;
    end
end
else begin
  case ch of
    cr : old;
    spc : new;
    bksp: rub;
  end;
end;
end;

procedure insertch;
var inspos: integer;
  newpos: integer;
begin
  if (l+wherex-xbeg > length (newdata)+1)
    then beep
  else if (wherex >= xbeg+field)
    then begin
      gotoxy (wherex-1,wherey);
      write (ch);
      newdata:= copy (newdata,1,length (newdata)-l)+ch;
      beep
    end
  else if (length (newdata) >= field)
    then beep
  else begin
    newpos:= wherex+l;
    inspos:= 1+wherex-xbeg;
    insert (ch,newdata,inspos);
    gotoxy (newpos-1,wherey);
    write (copy (newdata,inspos,80));
    gotoxy (newpos,wherey);
  end;
  exit:= false;
end;

procedure passcontrol;
begin
  control:= upcase (ch);
  if (newdata= '')
    then newdata:= olddata;
  exit:= true;
end;
begin \{data\_control\}
\beg:= xref;
ybeg:= line;
if (field= 0)
then xend:= xbeg
else xend:= xbeg+length(newdata);
yend:= line;
gotoxy (xend,yend);
repeat
getch;
if (field= 0)
then passcontrol
else
if (ch in ctrlset)
or scancode and
(ch in scanset)
then move
else
if not scancode and
(ch in dataset)
then insertch
else passcontrol;
until exit;
end; \{data\_control\}

The header contains five items. \textit{Olddata} contains a correct, previously entered value of the variable, and \textit{newdata} contains the new value currently being entered. This is necessary because error checking is performed in the middle editor after exit from data\_control. If \textit{newdata} is correct then it replaces \textit{olddata}. If \textit{newdata} is incorrect, a beep is heard, and it is returned to data\_control along with \textit{olddata} where it must be corrected or replaced with \textit{olddata} using the space bar (spc).

Upon entering data\_control, the cursor is placed at (xref,line). During character-wise editing and bulk erasing
the cursor is retained in a horizontal region beginning at xref and extending field spaces to the right.

During the course of editing, ch may be any of the characters shown in ctrlset or scanset. Distinction is made between ordinary control characters (ctrlset) and scan code (two byte) characters (scanset). Global boolean scancode (not passed) is set to true in the event that a scancode character is entered (see procedure getch).

If field is zero, only a control character is retrieved, and olddata and newdata can be dummy variables. Otherwise, retrieval of data and control information occurs as usual. Procedures backspace, forspace, rub, and deletech perform just as their names indicate. Insertion is the only mode of character entry (a beep is heard if the field is overfilled) and is accomplished by procedure insertch, which is executed whenever the entered character is not a control character (ctrlset,scanset) and is printable (dataset). Procedure new performs the bulk erase/olddata toggle. Procedure old is executed in response to cr, at which time data_control is exited with control containing a fictitious substitute character called out. The reason for the substitute will be apparent after study of the middle editor.

Looking at the last clause in the procedure, note that if a match is not found after comparing the entered control
character with the control sets internal to `data_control` and
the printable data set, then the control character will be
passed back (`passcontrol`) to the calling procedure in the
middle editor. In this case, the character is usually `uarr`
or `darr` for viewing/selecting values in the item's data
ring—a task that can only be performed by the appropriate
middle editor procedure.

In general, efforts are made to relegate function to
increasingly lower levels in order to centralize code and
impose operational uniformity; yet, termination conditions
on repeat loops permit traversal (same level) or ascendance
(next highest object-procedure level) to levels where the
entered control character has relevance and may generate the
appropriate external behavior. (Natural systems provide a
model for this kind of organization. For example, in
mammals, the lower-motor-neuron (LMN) system, cerebellum,
and motor cortex components of the motor control system have
for millions of years been proven efficient and reliable.
The reader of bionics may appreciate the parallels.) Thus,
an attempt is made in the editor to strike the most elegant
balance between low-level distributed application of
centralized code (`data_control` and the three, type-divided,
middle editor procedures) and high-level procedure-wise
localization of simple, distinct data objects (the many data
Section 3 - The Middle Editor

The middle editor, as mentioned, is the central control junction for the editor. It consists of procedures thename, thesymbol, and thequantity. Were it not for the rigid variable type specifications required in procedure headers, the three would be a single procedure. Procedure thesymbol is shown below. The profile and opcurve sections of the procedure have been omitted, leaving only the mode (control mode) sections for illustrative purposes.

```pascal
procedure thesymbol (var symbol : scalar;
  xref, line : integer;
  field : integer;
  symbolset : scalarset;
  var control: char);

var d: datastring;

procedure getsymbol;

var olddata: datastring;
  newdata: datastring;
  correct: boolean;
```
procedure checksymbol;

procedure checkmode (var mode: scalar;
    modeset: scalarset);

var newmode: scalar;

begin
    olddata:= cap (olddata);
    newdata:= cap (newdata);
    if modesymbol (newdata,newmode)
        and (newmode in modeset)
    then begin
        correct:= true;
        control:= ack;
        if (newdata<>olddata)
            then updatenode;
        mode:= newmode;
    end
    else begin
        correct:= false;
        beep;
    end;
end;

begin (checksymbol)
begin case symbol of
    P..PID : checkmode (symbol,symbolset);
    LIN..XPO : checkopcurve (symbol,symbolset);
    DESN..SINE: checkprofile (symbol,symbolset);
end;
end;
procedure showsymbols;

procedure showmodes (var mode: scalar;
                    modeset: scalarset);

const first= P;
    last = PID;

begin
    repeat
        case control of
            uarr: if (mode= last)
                then mode:= first
                else mode:= succ(mode);
            darr: if (mode= first)
                then mode:= last
                else mode:= pred(mode);
        end;
        until (mode in modeset);
    newdata:= modestring[mode];
    putmode (mode,xref,line,field);
end;

begin {showsymbols}
    case symbol of
        P..PID : showmodes (symbol,symbolset);
        LIN..XPO : showopcurves (symbol,symbolset);
        DESN..SINE: showprofiles (symbol,symbolset);
    end;
end;
begin {getsymbol}
  textcolor (cyan);
  putsymbol (symbol,xref,line,field);
  olddata:= symbolstring[symbol];
  newdata:= olddata;
  repeat
    data_control(olddata,newdata,xref,line,field,control);
    case control of
      out : checksymbol;
      uarr: showsymbols;
      darr: showsymbols;
    end;
    until (control in [out,ack,cr]) and correct;
  textcolor (yellow);
  putsymbol (symbol,xref,line,field);
end;

begin {thesymbol}
  repeat
    data_control (d,d,xref+field+1,line,0,control);
    case control of
      spc : getsymbol;
      larr: getsymbol;
      bksp: getsymbol;
    end;
    until (control in [ack,cr,home,endk,ccr,pgup,pgdn,uarr,darr,esc])
    or (control in menuset);
end;

Upon entering thesymbol, the cursor is placed to the right of the data such that there is one space between them. In the event that there are units associated with the data item (a quantity), the cursor will lie between the two data fields with one space on either side.

Pressing spc, larr, or bksp will execute getsymbol. Ignoring for now the first key or in the termination set, we
review the effect of pressing the remaining keys. Pressing CR or ENDK will send the cursor to the next item (moving down) whether in the same or following box. Pressing HOME will send the cursor to the previous item whether in the same or preceding box. Pressing CCR will send the cursor to the first item of the next box. PGUP and PGDN change the displayed page. UARR and DARR 'rotate' the ring a distance of one node under the current page template, changing the data in the boxes. ESC takes us out of Nodeview back to Ringview. Also, if the key pressed is in the menu set ('D' or 'S'), exit into the selected application occurs in this page.

Entering GETSYMBOL (entering the data field), the data is rewritten in light gray (often appears gray-blue), OLDDATA and NEWDATA are assigned the string equivalent of the data (a scalar in this case), and we enter DATA CONTROL. The cursor now hugs the right side of the data field, and the inner editor is operative. After editing and pressing CR, control is returned to GETSYMBOL where CHECKSYMBOL is executed.

CHECKSYMBOL verifies first that the entered string is in fact a mode (FUNCTION MODESYMBOL) and then checks for membership in MODESET to see if it may be entered in this particular instance. (Actually, there is only one control mode in the model. However, there are four different
instances of input profiles and four different instances of operating curves. In these cases, the editor may be configured to allow only certain scalar values to be entered for each profile and opcurve.)

If newmode is correct then boolean correct is set and the node's date is set to the current date (updatenode). At this point, control is set to ack. This is a fictitious key signifying to the calling item's box that it should acknowledge by checking to see if the new scalar (mode) differs from the old one. This check is made by getControlmode in the Controls box of pagel, for example. If the scalar is the same then the cursor returns to the same item. If the scalar is different then the lines under the scalar are erased (rubitem), a new set of underlying items and corresponding line positions are determined in accordance with pre-established boolean and vertical coordinate templates (newitem), and the new set of items is then placed in the box under the scalar (putitem). The cursor then returns to the same item (the scalar).

Note that checksymbol may be reconfigured so that control is set to cr, in which case the same above check will occur but the cursor will now move directly to the next item.

Finally, as a third option, assignment to control in checksymbol may be eliminated, leaving control containing out
(set during exit from data\_control). Now there will be no immediate return to the calling item's box. Instead, the cursor will be retained at the same item until the user presses cr (or another MOVE key). Execution will then proceed as per the preceding paragraph. The difference between this and the preceding case (control\:= cr vs. control\:= out) is that in the first case, a single stroke of cr effects an immediate, nearly simultaneous readjustment of box contents and cursor movement, whereas in the second case, two strokes of cr are required: the first to exit the inner editor and move the cursor a few spaces back to center; the second to effect readjustment of box contents and cursor movement.

The reader can now understand the reasons for a three-key termination condition on getsymbol. In summary, the three preceding paragraphs give information on how to change item-to-item cursor movement by simply changing the character assigned to control in checksymbol. Beyond this, no rewriting of the editor code is required.

\textbf{Showmodes} makes use of Pascal's pred and succ functions to traverse in a circular fashion a linear section of the global scalar type definition shown near the top of A.EDIT. To achieve circular traversal of control mode scalars, for example, local constants first and last identify the boundaries P and PID, respectively. We hereby fake the ring
traversal of a non-ring structure. Note that the same up and down arrow keys are used to traverse an item's data ring, but we must be in a data field (light gray), of course, for this to occur.

The other two middle editor procedures are the **name** and the **quantity**; they are structured in much the same way. The **name**, however, shows groups of allowable characters to the limit of the field, moving through the character set in the direction indicated by the arrow key. (The file name (field=12) and node name (field=50) are the only names in the editor.)

The **quantity** differs in that it shows min and max values for the number (adjusted as necessary for the chosen units). More importantly, an on-line units converter changes the number with each change of the units in order to maintain a constant quantity. This is accomplished by storing an array of multiplier conversion constants (global array factor) such that the constant associated with the default units is 1. Generally, conversion from one set of units to another is accomplished by first dividing by factor[units] and then multiplying by factor[newunits]. Also, boolean variable enforce is added to determine compliance with constant or variable min and max real number boundaries. If enforce is true, a beep is heard if the user tries to enter a number outside [min,max]; the cursor is bound to the item until the
user corrects the situation. If enforce is false, a beep is heard as a warning and the cursor is permitted to move to the next item. See Appendix A.EDIT for details on these two procedures.

Section 4 - The Item

The item procedure is the simplest of all object forms. A representative quantity item TI is shown below.

procedure itemTI (dowhat: message);
  const field= 8;
  min= 0.001;
  max= 999.990;
  timeset: scalarset= [sec,minutes,hrs];
  enforce= true;

begin
  with worknode^system do
  case dowhat of
    put_name: align ('TI= ',xref,line);
    put_data: putquantity (TI,UNITSti,xref,line,field);
    get_data: thequantity (TI,UNITSti,xref,line,field,
      min,max,timeset,enforce,control);

  end
end;

Procedures align and putquantity are very simple, too, and may be found in APPENDIX A.UTILITIES. Thequantity has already examined. Observe that the item procedure is simply a convenient, clean way of consolidating all specifications and operations that relate to a single data item, whether it be a name (string), symbol (mode, profile, and opcurve
scalars), or quantity (real number and units scalar). Such self-containment is essential to clarity in a large program.

Also, it is important to notice that the item procedure is completely meaningless if its case structure is not preceded by with worknode^.system do. Worknode always points in memory to the model record currently being viewed/edited/simulated.

Section 5 - The Box

The essentials of the Controls box of page 1 are shown below. The complete procedure is too bulky to include here (see APPENDIX A.EDIT).

procedure controls (dowhat: message);

const left= 24;
  xref= 40;
  right= 56;

  top= 5;
topleft= 7;
lines= 5;
bottom= 12;

var doALPHA: boolean;
doKC : boolean;
doTI : boolean;
doTD : boolean;

DISP: array[0..4] of integer;
procedure newitem;

procedure SETp;
begin
  doALPHA := false;
  doKC := true;
  doALPHA := false;
  doTD := false;
end;

procedure SETpi;
begin
  doALPHA := false;
  doKC := true;
  doTI := true;
  doTD := false;
end;

procedure SETpd;
begin
  doALPHA := true;
  doKC := true;
  doTI := false;
  doTD := true;
end;

procedure SETpid;
begin
  doALPHA := true;
  doKC := true;
  doTI := true;
  doTD := true;
end;

procedure DISPp;
begin
  DISP[0] := 0;
  DISP[2] := 1;
end;
procedure DISPpi;
begin
  DISP[0]:= 0;
  DISP[2]:= 1;
  DISP[3]:= 2;
end;

procedure DISPpd;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[4]:= 3;
end;

procedure DISPpid;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
  DISP[4]:= 4;
end;

begin
  with worknode^.system do begin
    case controlmode of
      P : SETp;
      PI : SETpi;
      PD : SETpd;
      PID: SETpid;
    end;
    case controlmode of
      P : DISPp;
      PI : DISPpi;
      PD : DISPpd;
      PID: DISPpid;
    end;
  end;
end;
procedure itemControlmode (dowhat: message);

const field= 3;
  modeset: scalarset= [P,PI,PD,PID];

begin
  with worknode^.system do
  case dowhat of
    put_name: align ('Control mode= ',xref,line);
    put_data: putsymbol (controlmode,xref,line,field);
    get_data: thesymbol (controlmode,xref,line,field,
                        modeset,control);
  end;
end;

procedure itemALPHA (dowhat: message);
begin
end;

procedure itemKC (dowhat: message);
begin
end;

procedure itemTI (dowhat: message);
begin
end;

procedure itemTD (dowhat: message);
begin
end;

procedure putbox;
begin
end;

procedure rubbox;
begin
end;
procedure putitem;
begin
end;

procedure rubitem;
begin
end;

procedure getdata;

procedure getControlmode;
var oldmode: scalar;
begin
with worknode^.system do begin
oldmode:= controlmode;
itemControlmode (get_data);
if (controlmode<>oldmode)
then begin
rubitem;
newitem;
putitem;
end;
end;
end;

begin
case control of
  cr : item:= 0;
  home: item:= 4;
  endk: item:= 0;
  uarr: item:= 0;
  darr: item:= 0;
end;
repeat
  line:= topline+DISP[item];
case item of
  0: getcontrolmode;
  1: if doALPHA then itemALPHA (get_data);
  2: if doKC then itemKC (get_data);
  3: if doTI then itemTI (get_data);
  4: if doTD then itemTD (get_data);
end;
case control of
  cr : item:= item+1;
  home: item:= item-1;
  endk: item:= item+1;
end;
until not (item in [0..4])
or (control in menuiset)
or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (controls)
  newitem;
case dowhat of
  put_box : putbox;
  rub_box : rubbox;
  put_item: putitem;
  rub_item: rubitem;
  get_data: getdata;
end
end;

Constants left, right, top, and bottom are absolute
textmode screen coordinates that determine the size of the box. They are used by putbox and rubbox. Constant xref is
the leftmost position is all item fields (all items fields
are left-justified). Constant topline is the first line in
the box on which data is written (Controlmode goes here).
Constant lines is not used anywhere but identifies the maximum number of lines that may be occupied.

The SET and DISP procedures under newitem are the boolean and positional templates referred to earlier. Newitem is executed upon entering Controls and thereafter each time Controlmode is changed (getControlmode). Only those data items whose booleans are true will appear in the box. Each item will be positioned at topline+DISP[item] as shown in getdata. Note that indices for items and their displacements start at zero.

We examine what happens during polling (page traversal). Upon entering getdata, the starting item is determined based on the last entered control key. All keys except one imply that we start box traversal at the first item. Home, however, implies that we are moving backward through the boxes; therefore, the cursor is placed on the last item in the box in preparation for upward traversal through the box. Also, the cursor must be placed on the first line with each node-to-node movement since Controlmode is the only item that is always present. Hence, we include the uarr and darr selectors and set item to zero.

A similar case structure in the polling loop serves to increment/decrement item after data-control retrieval and ascendance from the inner and middle editors. The resulting traversal of items in the box will continue until another
true boolean is encountered, at which point we again descend through the middle and inner editors for another data-control retrieval.

Traversal continues until we: 1) move past the top or bottom of the box; 2) press a menu key; 3) initiate a higher-order movement or return to Ringview.
CHAPTER 4 - THE SIMULATORS

Section 1 - Overview

The top line of the two line editor menu gives the user a choice of Design or Simulate. Two simulators, one an animation the other a columnar numerical output, are provided with the editor as a way of exercising the model and presenting system responses in a palatable, hopefully educational form. Access is gained to the simulation submenu by pressing 'S'. The user then selects Animation or Numbers.

No applications are provided under Design. However, other individuals will be making contributions/refinements to both sides of the menu. For example, a root locus application is being developed concurrently and when ready will be incorporated under Design.

The following comments with regard to units conversion refer to both simulators. The user may select any one of several different units for each item found in the three pages of each system node. In order to make all parameters compatible during exercise of the model, procedures UNITStocompute and UNITStoreturn are provided (see APPENDIX A.UTILITIES). UNITStocompute is executed at or near the top
of both simulators and converts all current record model parameters to the default set of units. Calculations then proceed directly in the default set. Reconversion back to the record's unit set is periodically required during output, and these conversions are performed locally in the set procedures of animation and in writedesignline and writecurrentline of numbers. Upon exit from either simulator, UNITStoreturn is executed, converting all current record model parameters back to the their original values. Note that the conversion constants are not inserted throughout the model itself because this severely degrades readability of the algebra.

Section 2 - Animation

Upon entering Animation, the screen shown in Photo 7 is seen. See APPENDIX A.ANIMATION for the listing. Three individually placed pictures (Turbo Pascal's getpic and putpic are employed) are present. The largest is the overall system including nearly everything but the the two three-sided tanks, which are drawn to scale (internally and relative to each other) immediately after putpicing system (drawtanks). The other two pictures are the output valve and the O2box (THETA2 box), both of which must must be able to change their vertical positions in accordance with H3, the height of the output pipe (drawoutpipe and drawO2box).
Photo 7. Screen during Animation.
To save time switching between the editor and the animation, these pictures are loaded (prepanimator) in the main body of the editor and stored globally instead of here in animation.

Behavior and code structure will be described in one pass, for the latter is not especially interesting. The code is well objectified but not thoroughly; thus, it does not appear as polished as that of the editor core and cannot appear so because of the inherently tedious nature of graphics operations. Pictures waterhor and waterver are used to fill in a static way all sections of water pipe. Flow simulation is minor in physical scope but is achieved, nonetheless, by the blinking on and off of small, one-raster-thick horizontal and two-column-thick vertical slices (waterh and waterv) at pipe-air and pipe-valve interfaces. The intended affect is one of turbulence or bubbling at the interfaces in the system's piping.

Full pipe flow simulation is not possible for reasons relating to speed not aesthetics. Indeed, very satisfactory, high-speed flow patterns were achieved during development in one small isolated section of pipe. However, there are ten sections of pipe to deal with if one counts right angles, devices, and tank walls as boundaries. Running a circuit in order to achieve piece-wise continuous, multiplexed flow proves to be jerky, slow, and ultimately unrealistic for circuits containing more than three or four
pieces of pipe. Additional overhead such as writing numbers to the boxes, raising tank liquid levels, and exercise of the model itself only worsen the situation. See setF1, setF2, setFO, and setFl.

Graphical simulation of rising tank liquid levels is straightforward. The Turbo-primitive draw command is used to draw horizontal lines from one tank wall to the other repeatedly over some incremental height developed during the last calculation time interval (setH1 and setH2).

In general, all of the set procedures consist of three parts: a first part to place the number in its box; a second part to make the appropriate graphical change (if appropriate); and a third part to place units in the boxes upon Reset (if appropriate).

Operation is simple. The user has a choice of seeing system Design values or Current values in the boxes (designvalues and currentvalues). At the outset, design values are shown in all the boxes and the time box (center) reads 00:00:00 (hrs:min:sec). Note that design values and current values are equal at time=0. Pressing 'C' when design values are showing shows current values. Oppositely, pressing 'D' when current values are showing shows design values. The two sets of values will toggle in the boxes as long as the key pressed does not correspond to what is showing. When the key pressed does correspond to what is
showing, the simulation is set into motion continuing from the current value of time. Pressing either key during simulation stops the simulation and shows the corresponding values.

Another useful feature is that any time during simulation, the user may Reset, causing the numbers in all the boxes and the tank levels to return to their design values. The clock is also reset to zero. Simulation may now proceed as usual.

Also, should the user indiscreetly choose the controller gain, tank diameters, and/or pump capacity, it is rather likely that an overflow will occur in one or both tanks. When this happens, the water level will remain at the top of the tank and 'OVERFLOW' will appear nearby, disappearing only when the level begins to drop.

Finally, pressing esc brings the user back into the editor at the same point from which he left. The frame, all boxes and their contents, and the last cursor position are re-established.
Section 3 - Numerical Output

Upon entering numbers, the screen shown in Photo 8 is seen. See APPENDIX A.NUMBERS for the listing. Although it leaves more to the imagination, this simulator is probably the more useful of the two because of its speed and other features.

The original frame is retained. The top box still contains the node header. The middle window contains eight variable headers (yellow) together with their respective units (light gray); the values of the variables are shown two lines below. The bottom box, as usual, contains a menu.

With respect to the viewing of design values and current values, the same scheme is employed. However, because this application operates in textmode, colors are used to clearly separate the two sets of values. Design values (see designvalues) appear in cyan and current values (currentvalues) in yellow.

The default mode of presentation places numbers repeatedly on the same line at each _t_. Pressing 'S' will light the word 'Scroll' in the menu and cause lines of numbers to print contiguously down the screen until the window is full, at which time the middle window is erased and printing continues again at the top. Scroll is a toggle.
Photo 10. Scrolling Numerical Output.
Photo 11. Change of Output Interval.
Interval (procedure interval) allows the user to determine the number of calculation time steps between each output. Pressing 'I' clears the bottom line of the menu and prompts the user for the desired number of time steps STEPS (see Photo 11). After entering a number (the protocol is the same as the editor's), the prompt is cleared and the bottom line of the menu is replaced. The base calculation interval for the model is 0.25 secs (0.0041666666 min).

The user is able to hardcopy any portion of what will appear on the screen by pressing 'H' (procedure hardcopy). Output will halt at the current values whenever Hardcopy is toggled to allow for inspection of the output and adjustment of the printer. See APPENDIX A.OUTPUT for sample numerical output.

At any time except during output or change of interval, the cursor will be resting on one of the variable headers. It may be moved from header to header in either direction using the left and right arrow keys. The user selects the variable he wants to appear in a given column by operating the up and down arrows in a thumbwheel fashion. Any of the following may be placed in each column: <nothing>, t, H1, H2, X3, X4, EH, EV, V2S, V2, VM, F1, F2, FO, H2S, THETA1, THETA2, FL. This means that the same variable may appear in more than one column. For example it might be useful to place t in the leftmost and rightmost columns to reduce eye
span. Also, any number of columns (even all of them) may be left blank, as desired.

Pressing 'R' will reset the model and place design conditions again on the top line below the headers. Pressing esc returns us to the editor.
SUMMARY AND CONCLUSIONS

The editor/simulator package is a friendly, educational tool designed to supplement classroom and laboratory instruction in a course on control theory or systems analysis.

The object of simulation is a PID two-tank level control system exhibiting at most fourth-order dynamics. Two simulators operating under the editor provide graphical animation and multi-column numerical output. In both cases, the user may stop simulation and quickly view the system's current state or design state. The numerical simulator provides also for hardcopy output, scrolling/non-scrolling output, a variable calculation interval, and an output speed several times greater than that of the animator.

The editor consists of a memory-resident ring or circular deque of system nodes (records), each containing all of the information necessary to support simulation. Each node is further divided into three display pages of boxed data. Editor topology is toroidal (node-to-node movement); and the toroid's crosssection is triangular (page-to-page movement). Traversal of the ring is accomplished then by node-to-node movement along any one of the toroid's three collateral sub-rings. Since each page format is fixed for
all nodes, Nodeview traversal requires only that the contents of the page's boxes be changed to reflect each node-to-node movement. This provides an easy way to quickly scan and compare large amounts of data on any page level.

Ringview traversal, on the other hand, enables the user to view the contents of the ring in abbreviated form, where each node is represented by a single line showing its 3-digit, quick-ID number, 50 character descriptive name, and the date of last editing. Ringview thus permits comprehensive inspection of very large rings (hundreds of nodes) in little time and few keystrokes. Switching between Ringview and Nodeview is accomplished by pressing cr (zoom in) and esc (zoom back).

The two-mode Ringview/Nodeview data scoping and toroidal topology together with a standard, uniform data entry protocol at the item level produce a very streamlined, economical, nearly closest-packed editor functionality, where economy (or distance) is measured essentially by the number of keystrokes required to move between any two points in the editor's data space (toroid).

Topological characterizations alone hint at the presence of (need for) object-oriented structure in the editor program source code. It is painfully obvious that the editor program should have been written in an object-
oriented language—Smalltalk would be an excellent choice. At this writing (July 1986) a few, fairly powerful implementations of this language exist for the microcomputer. Unfortunately, the language has hitherto been confined to minicomputers (and up) because of large memory requirements.

Insofar as objectified Pascal procedural structure is concerned, one may argue for the program's English-like readablility. However, after thorough engrossment in the editor's code, one becomes aware of subtly vulnerabilities related mainly to scope of variables. Also, the passing of message selectors can only appear so graceful in Pascal, for the language was not designed to be used in this manner. Few modifications to the program should be attempted beyond the obvious reformatting of pages and their boxes. I recommend a first rewriting in Modula 2, if necessary, and then in Smalltalk. However, the editor concept should be clear enough to attempt the Smalltalk implementation immediately.

Improvements: A block masking operation is needed in Ringview. At present, the user must mask each node individually using del. The home and end keys might be used to place the begin-end markers and del to color in the red and effect the mask. Also, as mentioned, the graphical animator falls short of the arcade-game sophistication and
entertainment value everyone expects today. The flow-
graphical routines needed to be improved visually and
written in machine language. More vertical resolution for
the sake of realistic level dynamics would be nice, too.
## APPENDIX

**A. OUTPUT - Sample Numerical Output**

<table>
<thead>
<tr>
<th>t (min)</th>
<th>H1 (ft)</th>
<th>H2 (ft)</th>
<th>H2S (ft)</th>
<th>VM (v)</th>
<th>F1 (cfm)</th>
<th>F2 (cfm)</th>
<th>F0 (cfm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>20.201</td>
<td>20.000</td>
<td>20.000</td>
<td>0.359</td>
<td>35.902</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>0.250</td>
<td>20.201</td>
<td>20.000</td>
<td>20.000</td>
<td>0.359</td>
<td>35.902</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>0.500</td>
<td>20.201</td>
<td>20.000</td>
<td>20.000</td>
<td>0.359</td>
<td>35.902</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>0.750</td>
<td>20.201</td>
<td>20.000</td>
<td>20.000</td>
<td>0.359</td>
<td>35.902</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>1.000</td>
<td>20.201</td>
<td>20.000</td>
<td>20.000</td>
<td>0.359</td>
<td>35.902</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>1.250</td>
<td>32.985</td>
<td>20.758</td>
<td>30.000</td>
<td>7.303</td>
<td>730.298</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>1.500</td>
<td>42.909</td>
<td>22.194</td>
<td>30.000</td>
<td>7.061</td>
<td>706.126</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>1.750</td>
<td>50.204</td>
<td>23.937</td>
<td>30.000</td>
<td>6.481</td>
<td>648.069</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>2.000</td>
<td>54.890</td>
<td>25.836</td>
<td>30.000</td>
<td>5.644</td>
<td>564.422</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>2.250</td>
<td>56.961</td>
<td>27.779</td>
<td>30.000</td>
<td>4.618</td>
<td>461.762</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>2.500</td>
<td>56.504</td>
<td>29.670</td>
<td>30.000</td>
<td>3.465</td>
<td>346.533</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>2.750</td>
<td>53.751</td>
<td>31.417</td>
<td>30.000</td>
<td>2.257</td>
<td>225.664</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>3.000</td>
<td>49.127</td>
<td>32.926</td>
<td>30.000</td>
<td>1.070</td>
<td>106.979</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>3.250</td>
<td>43.322</td>
<td>34.094</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>3.500</td>
<td>38.366</td>
<td>34.818</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>3.750</td>
<td>35.684</td>
<td>35.085</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>4.000</td>
<td>34.970</td>
<td>34.957</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>4.250</td>
<td>34.746</td>
<td>34.733</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>4.500</td>
<td>34.523</td>
<td>34.510</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>4.750</td>
<td>34.300</td>
<td>34.288</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>5.000</td>
<td>34.079</td>
<td>34.067</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>5.250</td>
<td>33.859</td>
<td>33.847</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>5.500</td>
<td>33.639</td>
<td>33.628</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>5.750</td>
<td>33.421</td>
<td>33.409</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>6.000</td>
<td>33.204</td>
<td>33.192</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>6.250</td>
<td>32.987</td>
<td>32.976</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>6.500</td>
<td>32.772</td>
<td>32.761</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>6.750</td>
<td>32.558</td>
<td>32.546</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>7.000</td>
<td>32.344</td>
<td>32.333</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>7.250</td>
<td>32.132</td>
<td>32.121</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>7.500</td>
<td>31.920</td>
<td>31.909</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>7.750</td>
<td>31.710</td>
<td>31.699</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>8.000</td>
<td>31.500</td>
<td>31.489</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>8.250</td>
<td>31.292</td>
<td>31.281</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>8.500</td>
<td>31.084</td>
<td>31.074</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>8.750</td>
<td>30.878</td>
<td>30.867</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>9.000</td>
<td>30.672</td>
<td>30.661</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td>9.250</td>
<td>30.467</td>
<td>30.457</td>
<td>30.000</td>
<td>0.000</td>
<td>0.000</td>
<td>35.902</td>
<td>55.902</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30.264</td>
<td>30.061</td>
<td>29.859</td>
<td>29.658</td>
<td>29.458</td>
<td>29.259</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30.253</td>
<td>30.051</td>
<td>29.849</td>
<td>29.648</td>
<td>29.448</td>
<td>29.250</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30.000</td>
<td>30.000</td>
<td>30.000</td>
<td>30.000</td>
<td>30.000</td>
<td>30.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8.129</td>
<td>8.090</td>
<td>8.052</td>
<td>8.014</td>
<td>7.975</td>
<td>7.937</td>
<td></td>
</tr>
<tr>
<td></td>
<td>68.754</td>
<td>68.523</td>
<td>68.293</td>
<td>68.063</td>
<td>67.833</td>
<td>67.604</td>
<td></td>
</tr>
</tbody>
</table>

This appears to be a table with values in the first three columns and possibly some calculations or measurements in the remaining columns. The specific meaning or context of the data is not clear from the image alone.
program twotanks (input, output);

  type intpoint = ^integer;
  charset = set of char;
  nodestring = string[31];
  datastring = string[51];
  datestring = string[103];
  scalar = (LIN, XPO,
            PI, PD, PID,
            DESN, STEP, PULS, RAMP, SINE,
            sec, minutes, hrs,
            inches, ft, cm, m,
            sqin, sqft, sqcm, sqm,
            cfs, cfm, cfh, lps, lpm, lph,
            ft_sec, ft_min,
            cps, cpm, cph,
            mv, v,
            percent,
            none);
  scalarset = set of scalar;
  symbolarray = array[scalar] of string[12];
  curvearray = array[LIN..XPO] of string[3];
  modearray = array[PI..PID] of string[3];
  profilearray = array[DESN..SINE] of string[4];
  unitarray = array[sec..none] of string[12];
  factorarray = array[sec..none] of real;
  fieldarray = array[sec..none] of integer;
  message = (put_box, rub_box, put_name, rub_name,
              put_data, rub_data, get_data, put_item, rub_item);

  twotanksystem = record
    number: nodestring;
    name: datastring;
    date: datestring;
    controlmode: scalar;
\begin{verbatim}
ALPHA, 
KC, 
TI, 
TD: real;

UNITSa1pha, 
UNITSkc, 
UNITSti, 
UNITStd: scalar;

HT1, 
A1, 
HT2, 
A2, 
H3: real;

UNITSh1, 
UNITSa1, 
UNITSh2, 
UNITSa2, 
UNITSh3: scalar;

H2SB, 
THETA1B, 
THETA2B, 
FLB: real;

UNITSh2s, 
UNITSth1, 
UNITStheta1, 
UNITStheta2, 
UNITStf1: scalar;

PROFILEh2s, 
PROFILEtheta1, 
PROFILEtheta2, 
PROFILEf1: scalar;

tOh2s, 
tih2s, 
SPh2s, 
SPIh2s, 
SP2h2s, 
Wh2s: real;

UNITStOh2s, 
UNITStih2s,
\end{verbatim}
UNITSsp2h2s,
UNITSsp1h2s,
UNITSsp2h2s,
UNITSwh2s: scalar;

t0theta1,
t1theta1,
SPtheta1,
SPitheta1,
SP2theta1,
Wthetal: real;

UNITSt0theta1,
UNITStltheta1,
UNITSpstheta1,
UNITSpitheta1,
UNITSp2theta1,
UNITSwthetal: scalar;

t0theta2,
t1theta2,
SPtheta2,
SPitheta2,
SP2theta2,
Wthetali: real;

UNITSt0theta2,
UNITStltheta2,
UNITSpstheta2,
UNITSpitheta2,
UNITSp2theta2,
UNITSwthetali: scalar;

t0f1,
t1f1,
SPf1,
SP1f1,
SP2f1,
Wf1: real;

UNITSt0f1,
UNITStf1,
UNITSpf1,
UNITSpf1,
UNITSp2f1,
UNITSwf1: scalar;
OPCURVEvalve1,
OPCURVEvalve2,
OPCURVetrans2,
OPCURVEpump: scalar;

CV1,
CV2: real;

UNITScv1,
UNITScv2: scalar;

B1,
B2,
BT,
BM: real;

UNITSb1,
UNITSb2,
UNITSbt,
UNITSbm: scalar;

H2MIN,
H2MAX,
V2MIN,
V2MAX: real;

UNITSh2min,
UNITSh2max,
UNITSv2min,
UNITSv2max: scalar;

VMIN,
VMAX,
F1MIN,
F1MAX: real;

UNITSvmin,
UNITSvmax,
UNITSf1min,
UNITSf1max: scalar;
end;

node= ^systemrecord;
systemrecord= record
deleted: boolean;
system: twotanksystem;
back: node;
fore: node;
end;

Rringheader= record
  number: nodestring;
  name: datastring;
  date: datestring;
end;

headernode= ^headerrecord;
headerrecord= record
  header: Rringheader;
  back: nodef;
  fore: node;
end;

(These five types refer to the animator)

systempicture= array[1..16480] of byte;
valvepicture= array[1..94] of byte;
boxpicture= array[1..138] of byte;
verslice= array[1..9] of byte;
horSlice= array[1..7] of byte;

const cr = #13;
ccr = #10;
out = #05;
ack = #06;
spc = #32;
bksp = #08;
ins = #82;
del = #83;
larr #75;
rrr = #77;
arr = #72;
darr = #80;
home = #71;
endk = #79;
pgup = #73;
pdgm = #81;
esc = #27;
null = #00;

crlset: charset= [cr,spc,bksp,esc];
moveset: charset= [esc,uarr,darr,pgup,pdgn];
dataset: charset= ['!''....'''];

opcurvestring: curvearray = ('LIN', 'XPO');
modestring: modearray = ('P', 'PI', 'PD', 'PID');
profilestring: profilearray = ('DESN', 'STEP', 'PULS', 'RAMP', 'SINE');
unitstring: unitarray = ('sec', 'min', 'hrs', 'in', 'ft', 'cm', 'm', 'sqin', 'sqft', 'sqcm', 'sqm', 'cfs', 'cfm', 'cfh', 'lps', 'lpm', 'lph', 'ft^3/2/sec', 'ft^3/2/min', 'cps', 'cpm', 'cph', 'mv', 'v', 'Z', '');

unitfield: fieldarray = (3, 3, 3, 2, 2, 2, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 12, 12, 3, 3, 3, 2, 2, 1, 1, 0, 1);
factor: factorarray = (60, 1, 0.0166666666667, 12, 1, 30.48, 0.3048, 144, 1, 929.0304, 0.09290304, 0.0166666666667, 1, 60, 0.471947, 28.31682, 1699.0092, 0.0166666666667, 1, 0.0166666666667, 1, 60, 1000, 1, 1, 1);

h = 0.00416666666666; (1/4 second in minutes)

($I$ defsys.inc)
var cleanheap: intpoint;
mustsave: boolean;
control: char;
scancode: boolean;

filename: datastring;
nodename: datastring;
headnode: node;
worknode: node;
setring: boolean;
setpage: boolean;

page: integer;
menuset: charset;

t: real;

(RUNNING MODEL VARIABLES)

H2S,
THETA1,
THETA2,
FL: real;

H1,
H2,
x3,
x4: real;
EH: real;

H1B,
H2B,
x3B,
x4B: real;
EHB: real;

UNITSh1,
UNITSh2,
UNITSx3,
UNITSx4,
UNITSh: scalar;

V2S,
V2,
EV,
VM,
F1,
F2, 
FD: real;

V2SB, 
V2B, 
EVB, 
VMB, 
F1B, 
F2B, 
FOB: real;

UNITSv2s, 
UNITSv2, 
UNITSev, 
UNITSv1, 
UNITSf1, 
UNITSf2, 
UNITSfo: scalar;

UNITSf1: scalar;

F2MAX, {The max of FMAX}
FOMAX, (and these three)
FLMAX, {is FMAX}
FMAX: real; {FMAX used as scale for interpolation}

(These five variables refer to the animator)

system: systempicture;
valve: valvepicture;
02box: boxpicture;
waterh: horslice;
waterv: verslice;

{$1 utility.inc}

{$1 math.inc}

{$1 data_control.inc}

procedure thename (var name: datastrign;
                 xref,line: integer;
                 field: integer;
                 nameset: charset;
                 var control: char);
var d: datastring;

procedure getname;
var olddata: datastring;
newdata: datastring;
correct: boolean;

procedure checkname;
var n: integer;
size: integer;
oldname: datastring;
newname: datastring;

begin
oldname:= olddata;
newname:= newdata;
size:= length (newname);
correct:= true;
n:=1;
repeat
  if not (newname[n] in nameset) then begin
    correct:= false;
    beep;
  end;
  n:= n+1;
until not correct or (n= size+1);
if correct then begin
  control:= ack;
  if (newname<>oldname) then updatenode;
  name:= newname;
end;
end;

procedure showchars;
var ch: char;
  n: integer;

procedure nextch;
begin
  case control of
    uarr: ch:= succ(ch);
    darr: ch:= pred(ch);
  end;
end;

begin
  case control of
    uarr: ch:= copy (name,length(name),1);
    darr: ch:= copy (name,1,1);
  end;

  n:= 0;
  name:= '"';
  repeat
    nextch;
    while not (ch in nameset)
      do nextch;
    case control of
      uarr: name:= name+ch;
      darr: name:= ch+name;
    end;
    n:= n+1;
  until (n= field);
  newdata:= name;
  putname (name,xref,line,field);
end;

begin {getname}
  textcolor (cyan);
  putname (name,xref,line,field);
  olddata:= name;
  newdata:= olddata;
  repeat
    data_control (olddata,newdata,xref,line,field,control);
    case control of
      out : checkname;
      uarr: showchars;
      darr: showchars;
    end;
end;
until (control in [out,ack,cr]) and correct;
textcolor (yellow);
putname (name,xref,line,field);
end;

begin (thename)
repeat
  data_control (d,d,xref+field+1,line,0,control);
case control of
  spc : getname;
larr: getname;
bksp: getname;
end;
until (control in [ack,cr,home,endk,ccr,pdup,pdnd,uarr,darr,esc])
or (control in menuset);
end;

procedure thesymbol (var symbol : scalar;
  xref,line : integer;
  field : integer;
  symbolset : scalarset;
  var control: char );

var d: datastring;

procedure getsymbol;

var olddata: datastring;
  newdata: datastring;
  correct: boolean;

procedure checksymbol;

procedure checkmode (var mode: scalar; modeset: scalarset);

var oldmode: scalar;
  newmode: scalar;

begin
  olddata:= cap (olddata);
  newdata:= cap (newdata);
  if modesymbol (newdata,newmode) and (newmode in modeset)
then begin
  correct:= true;
  control:= ack;
  if (newdata<>olddata)
    then updatenode;
  mode:= newmode;
end
else begin
  correct:= false;
  beep;
end;
end;

procedure checkopcurve (var opcurve: scalar; opcurveset: scalarset);
var oldopcurve: scalar;
  newopcurve: scalar;
begin
  olddata:= cap (olddata);
  newdata:= cap (newdata);
  if opcurvesymbol (newdata,newopcurve) and (newopcurve in opcurveset)
    then begin
      correct:= true;
      control:= ack;
      if (newdata<>olddata)
        then updatenode;
      opcurve:= newopcurve;
    end
  else begin
      correct:= false;
      beep;
    end;
end;

procedure checkprofile (var profile: scalar; profileset: scalarset);
var oldprofile: scalar;
  newprofile: scalar;
begin
  olddata:= cap (olddata);
  newdata:= cap (newdata);
  if profilesymbol (newdata,newprofile) and (newprofile in profileset)
    then begin
      correct:= true;
      control:= ack;
      if (newdata<>olddata)
        then updatenode;
    end
  else begin
      correct:= false;
      beep;
    end;
end;
then updatenode;
  profile:= newprofile;
end
else begin
  correct:= false;
  beep;
end;
end;

begin (checksymbol)
  case symbol of
    P..PID : checkmode (symbol,symbolset);
    LIN..XPO : checkopcurve (symbol,symbolset);
    DESN..SINE: checkprofile (symbol,symbolset);
  end;
end;

procedure showsymbols;

procedure showmodes (var mode: scalar; modeset: scalarset);
  const first= P;
    last = PID;
begin
  repeat
    case control of
      uarr: if (mode= last)
          then mode:= first
          else mode:= succ(mode);
      darr: if (mode= first)
          then mode:= last
          else mode:= pred(mode);
    end;
  until (mode in modeset);
  newdata:= modestring(mode);
  putmode (mode,xref,line,field);
end;

procedure showopcurves (var opcurve: scalar; opcurveset: scalarset);
  const first= LIN;
last = XPO;
begin
  repeat
    case control of
      uarr: if (opcurve = last)
          then opcurve := first
          else opcurve := succ(opcurve);
      darr: if (opcurve = first)
          then opcurve := last
          else opcurve := pred(opcurve);
    end;
  until (opcurve in opcurveset);
  newdata := opcurvestring[opcurve];
  putopcurve (opcurve, xref, line, field);
end;

procedure showprofiles (var profile: scalar; profileset: scalarset);
begin
  repeat
    case control of
      uarr: if (profile = last)
          then profile := first
          else profile := succ(profile);
      darr: if (profile = first)
          then profile := last
          else profile := pred(profile);
    end;
  until (profile in profileset);
  newdata := profilestring[profile];
  putprofile (profile, xref, line, field);
end;

begin {showsymbols}
  case symbol of
    P..PID : showmodes (symbol, symbolset);
    LIN..XPO : showopcurves (symbol, symbolset);
    DESN..SINE: showprofiles (symbol, symbolset);
  end;
end;
begin {getsymbol}
  textcolor (cyan);
  putsymbol (symbol,xref,line,field);
  olddata:= symbolstring[symbol];
  newdata:= olddata;
  repeat
    data_control (olddata,newdata,xref,line,field,control);
    case control of
      out: checksymbol;
      uarr: showsymbols;
      darr: showsymbols;
    end;
  until (control in [out,ack,cr]) and correct;
  textcolor (yellow);
  putsymbol (symbol,xref,line,field);
end;

begin {thesymbol}
  repeat
    data_control (d,d,xref+field+1,line,0,control);
    case control of
      spc: getsymbol;
      larr: getsymbol;
      bksp: getsymbol;
    end;
  until (control in [ack,cr,home,endk,ccr,pgup,pgdn,uarr,darr,esc])
    or (control in menuset);
end;

procedure thequantity (var number : real;
  var units : scalar;
  xref,line : integer;
  field : integer;
  min,max : real;
  unitset : scalarset;
  enforce : boolean;
  var control: char );

var olddata: datastring;
newdata: datastring;
correct: boolean;
d : datastring;

procedure getnumber;

procedure checknumber;

var newnumber: real;

begin
  if realnumber (newdata,newnumber)
  then begin
    if (newnumber >= min) and (newnumber <= max)
    then correct:= true
    else begin
      if enforce
      then correct:= false
      else begin
        if (newnumber > max)
        then begin
          correct:= true;
          beep;
        end
        else correct:= false;
      end;
    end;
  end
  else correct:= false;
  if correct
  then begin
    control:= ack;
    if (newdata<>olddata)
    then updatenode;
    number:= newnumber;
  end
  else beep
end;

procedure shownumbers;

begin
  case control of
    uarr: if (number= min) then number:= max else number:= min;
    darr: if (number= max) then number:= min else number:= max;
  end
begin (getnumber)
  textcolor (cyan);
  putnumber (number,xref,line,field);
  olddata:= numberstring (number,field);
  newdata:= olddata;
  repeat
    data_control (olddata,newdata,xref,line,field,control);
    case control of
      out : checknumber;
      uarr: shownumbers;
      darr: shownumbers;
      end;
  until (control in [out,ack,cr]) and correct;
  textcolor (yellow!);
  putnumber (number,xref,line,field);
end;

procedure getunits;

var uxref: integer;
  ufield: integer;

procedure checkunits;

var newunits: scalar;

begin
  olddata:= cap (olddata);
  newdata:= cap (newdata);
  if unitsymbol (newdata,newunits) and (newunits in unitset) then begin
    correct:= true;
    control:= ack;
    if (newdata<>olddata) then updatenode;
    numbers:= number/factor[units]*factor[newunits];
    min:= min/factor[units]*factor[newunits];
    max:= max/factor[units]*factor[newunits];
    units:= newunits;
  end
else begin
  correct := false;
  beep;
end;
end;

procedure showunits;
const first := sec;
  last := none;
var oldunits: scalar;
begin
  oldunits := units;
  repeat
    case control of
      uarr: if (units = last) then units := first else units := succ(units);
      darr: if (units = first) then units := last else units := pred(units);
    end
  until (units in unitset);
  newdata := unitstring[units];
  number := number / factor[oldunits] * factor[units];
  min := min / factor[oldunits] * factor[units];
  max := max / factor[oldunits] * factor[units];
  putunits (units,uxref,line,ufield);
  putnumber (number,xref,line,field);
end;

begin (getunits)
  uxref := xref + field + 3;
  ufield := unitfield[units];
  textcolor (cyan);
  putunits (units,uxref,line,ufield);
  olddata := unitstring[units];
  newdata := olddata;
  repeat
    data_control (olddata,newdata,uxref,line,ufield,control);
    case control of
      out : checkunits;
      uarr: showunits;
      darr: showunits;
    end;
  until (control in [out,ack,cr]) and correct;
begin {thequantity}
min := \text{min} \times \text{factor}[\text{units}];
max := \text{max} \times \text{factor}[\text{units}];
repeat
\text{data_control} (d,d,xref+field+1,line,O,\text{control});
case \text{control} of
spc: \text{getnumber};
larr: \text{getnumber};
bksp: \text{getnumber};
'U': \text{if} (\text{units}<>\text{none}) \text{then} \text{getunits};
rarr: \text{if} (\text{units}<>\text{none}) \text{then} \text{getunits};
end;
until (\text{control} in [\text{ack},\text{cr},\text{home},\text{endk},\text{ccr},\text{pgup},\text{pgdn},\text{uarr},\text{darr},\text{esc}])
or (\text{control} in \text{menuset});
end;

procedure bigwindow (a,b,c,d: integer);
begin
\text{window} (a,b,c,d);
\text{CloneCodeSegment} (\text{TurboRunDataStart},\text{TurboRunDataLength});
end;

{APPLICATION PROCEDURES GO HERE}

{**X**} (array optimization)
{$!\jmd\graphix\typedef.sys}$
{$!\jmd\graphix\graphix.sys}$
{$!\jmd\graphix\kernel.sys}$
{$!\jmd\graphix\windows.sys}$
{$!\jmd\graphix\findwrld.hgh}$
{$!\jmd\graphix\modaxis.hgh}$
{$!\jmd\graphix\polygon.hgh}$

{$!\jmd\rl.pas}$

procedure design;
begin
{John Decatrel's root locus}
repeat until keypressed;
end;

procedure simulate;
var control: char;
    d: datastring;

procedure menufooter (dowhat: message);
const left= 2;
    right= 79;

    top= 22;
    bottom= 25;

procedure putitem;
const wordmenu= 'Numbers Animation';
begin
    center (top+1,left,right,wordmenu);
    writeyellowtips (wordmenu);
end;

procedure rubitem;
begin
    bigwindow (left+1,top+1,right-1,bottom-1);
    clrscr;
    bigwindow (1,1,80,25);
end;

begin
    bigwindow (1,1,80,25);
textcolor (yellow);
    case dowhat of
        put_item: putitem;
        rub_item: rubitem;
    end;
end; {menufooter}
begin
  menufooter (put_item);
  data_control (d,d,wherex,wherey,0,control);
  case control of
    'A': animator;
    'N': numbers;
  end;
  menufooter (rub_item);
end;

procedure thisfile;

  const defaultfilename= 'SYSTEM.SIM';

  left= 2;
  middle= 40;
  right= 79;

  topline= 2;

  var winleft: integer;
  winxref: integer;
  winright: integer;

  wintop: integer;
  winlines: integer;
  winbottom: integer;

  nodecount: integer;
  maskcount: integer;
  nodenumber: nodestring;

  box : array[1..4] of integer;
  item: array[1..4] of integer;

procedure windowspecs;

  const left= 2;
  right= 79;
top = 4;
bottom = 21;

begin
  winleft := left + 2;
  winxref := 7;
  winright := right - 1;
  wintop := top;
  winbottom := bottom;
  winlines := 1 + winbottom - wintop;
end;

procedure ringwindow;
begin
  bigwindow (winleft, wintop, winright, winbottom);
end;

procedure rubring;
begin
  ringwindow;
  clrscr;
  bigwindow (1, 1, 80, 25);
end;

function fileondisk (filename: datastring): boolean;
var testfile: file of twotanksystem;
begin
  assign (testfile, filename);
  {$I-}
  reset (testfile);
  {$I+}
  fileondisk := (IOresult = 0);
  close (testfile)
end;

procedure menufooter (dowhat: message);
const left = 2;
  right = 79;
  top = 22;
procedure putitem;

const wordmenu= 'Work file  Save  Renumber';

y= true;
n= false;

var keymenu: string[80];

begin
keymenu:=
'esc ins del pgup pgdn '+#024 '+' '+#025 ' cr spc';
center (top+1,left,right,wordmenu);
writeyellowtips (wordmenu);
writecenter (top+2,left,right,keymenu);
end;

procedure rubitem;

begin
bigwindow (left+1,top+1,right-1,bottom-1);
clrscr;
bigwindow (1,1,80,25);
end;

begin
bigwindow (1,1,80,25);
textcolor (yellow);
case dowhat of

put_item: putitem;
rub_item: rubitem;

end;
end; {menufooter}

function ringname: datastring;

const namesize= 12;

namefield= 51;
spc40= '

var space: integer;
spac•odd: boolean;
left: integer;
right: integer;
spcleft: string[40];
spcright: string[40];

begin
  space:= namefield-12-length (filename);
  spaceodd:= odd (space);
  left:= trunc (space/2);
  if spaceodd
    then right:= left+1
    else right:= left;
  spcleft:= copy (spc40,1,left);
  spcright:= copy (spc40,1,right);
  ringname:= 'HEADER'+spcleft+filename+spcright+'HEADER';
end;

procedure ringheader (dowhat: message);

const left= 2;
xref= 21;
middle= 40;
right= 79;

top= 1;
topline= 2;
lines= 1;
bottom= 3;

var line: integer;

function nameok: boolean;

var dotpos:integer;
suffix: datastring;

begin
  filename:= cap (filename);
dotpos:= pos ('.',filename);
  if (dotpos <> 0)
    then begin
      suffix:= copy (filename,dotpos,4);
      if (suffix= '.SIM') or (suffix= '.BAC')
        then nameok:= true
      else begin
        nameok:= false;
        beep;
      end;
textcolor (lightgray);
writecenter (topline,middle,right,'.SIM or .BAC suffix only');
delay (1500);
textcolor (black);
writecenter (topline,middle,right,'.SIM or .BAC suffix only');
textcolor (yellow);
end;
end
else begin
 filename:= copy (filename,1,8);
 filename:= concat (filename,'.SIM');
 nameok:= true;
end;
end;

procedure itemFilename (dowhat: message);

const field= 12;
 nameset: charset= ['!'..'^'];

begin
 case dowhat of
 put_name: align ('Work file name:',xref,line);
 put_data: putname (filename,xref,line,field);
 get_data: thename (filename,xref,line,field,nameset,control);
 end
end;

procedure putitem;

var item: integer;

begin
 item:= 0;
 repeat
 line:= topline+item;
 case item of
 0: begin
 itemfilename (put_name);
 itemfilename (put_data);
 end;

 end;
 item:= item+1;
 until (item>0);

procedure rubitem;

begin
  bigwindow (left+2,topline,right-2,topline+1);
  gotoxy (1,1);
  c1reol;
  bigwindow (1,1,80,25);
  end;

procedure putdata;
var item: integer;

begin
  item:= 0;
  repeat
    gotoxy (xref,topline+item);
    case item of
      0: itemfilename (put_data);
    end;

    item:= item+1;
  until (item)=0;
end;

procedure rubdata;

begin
  bigwindow (xref,topline,right-2,topline+1);
  gotoxy (1,1);
  c1reol;
  bigwindow (1,1,80,25);
  end;

procedure getdata;

var item: integer;

begin
  item:= 0;
  repeat
    line:= topline+item;
    case item of
begin
  bigwindow (1,1,80,25);
  textcolor (yellow);
  case dowhat of
    put_item: putitem;
    rub_item: rubitem;
    put_data: putdata;
    rub_data: rubdata;
    get_data: getdata;
  end;
end;  {ringheader}

procedure makenullring;

begin (makenullring)
  new (headnode);
  with headnode^ do begin
    "itemfilename (get_data);
end;
if not nameok then control:= null;
case control of
  cr : item:= item+1;
  home: item:= item-1;
end: item:= item+1;
end;
until not (item in [0]) or (control in [cr,ack]);
end;

procedure makenullring;

begin (makenullring)
  new (headnode);
  with headnode^ do begin
    "itemfilename (get_data);
end;
if not nameok then control:= null;
case control of
  cr : item:= item+1;
  home: item:= item-1;
end: item:= item+1;
end;
until not (item in [0]) or (control in [cr,ack]);
end;

begin
  bigwindow (1,1,80,25);
  textcolor (yellow);
  case dowhat of
    put_item: putitem;
    rub_item: rubitem;
    put_data: putdata;
    rub_data: rubdata;
    get_data: getdata;
  end;
end;  {ringheader}
delected:= false;
with system do begin
  number:= '000';
  name:= ringname;
  dates:= currentdate;
end;
fore:= headnode;
back:= headnode;
end;
worknode:= headnode;
nodecount:= 1;
nodenumner:= '001';
end;

procedure loadheadnode;

var nullsystem: twotanksystem;

begin
  read (ringfile, nullsystem);
  new (headnode);
  with headnode do begin
    deleted:= false;
    system:= nullsystem;
    system.name:= ringname;
    fore:= headnode;
    back:= headnode;
    end;
  end;
end;

procedure insertnode (thisnode: node; newsystem: twotanksystem);

var newnode: node;

begin
  new (newnode);
  with newnode do begin
    deleted:= false;
    system:= newsystem;
    back:= thisnode.back;
    fore:= thisnode;
    end;
  thisnode.back.fore:= newnode;
  thisnode.back:= newnode;
end; {insertnode}
procedure makeit;
begin
  rubring;
  textcolor(lightgray);
  writecenter(topline,middle,right,'Loading');
  assign(ringfile, filename);
  reset(ringfile);
  loadheadnode;
  nodecount:= 1;
  nodenumber:= '001';
  repeat
    read(ringfile, newsystem);
    insertnode(headnode, newsystem);
    if (newsystem.number > nodenumber)
      then nodenumber:= newsystem.number;
    nodecount:= nodecount+1;
    until eof(ringfile);
  close(ringfile);
  nodenumber:= incstr(nodenumber);
  textcolor(black);
  writecenter(topline,middle,right,'Loading');
  textcolor(yellow);
  worknode:= headnode;
end;

procedure saynewfile;
begin
  rubring;
  textcolor(lightgray);
  writecenter(topline,middle,right,'New file');
  delay(1000);
  textcolor(black);
  writecenter(topline,middle,right,'New file');
  textcolor(yellow);
end;

procedure saynotsaved;
begin
  textcolor(lightgray);
  writecenter(topline,middle,right,filename+' not saved');
  delay(1000);
  textcolor(black);
  writecenter(topline,middle,right,filename+' not saved');
textcolor (yellow);
end;

begin (makering)
bigwindow (1,1,80,25);
if mustsave
then saynotsaved
else begin
  ringheader (get_data);
  ringheader (put_data);
  mark (cleanheap);
  maskcount:= 0;
  if fileon_disk (filename)
  then makeit
  else begin
    saynewfile;
    makenullring;
  end;
  mustsave:= false;
  setring:= true;
  setpage:= true;
end;
end;

procedure savering;
const left= 2;
  middle= 40;
  right= 79;
  topline= 2;
var ringfile: file of twotanksystem;
  system: twotanksystem;
  pointer: node;

procedure checknullring;
var holdcount: integer;
begin
  if (nodecount-maskcount > 1)
  then mustsave:= true
  else begin
    textcolor (lightgray);
    writecenter (topline,middle,right,'File is empty');
    delay (10000);
procedure askoverwrite;

var ch: char;

begin
  textcolor (lightgray);
  writecenter (topline,middle,right,'Overwrite old '+filename+'? (Y/N)');
  repeat
    read (kbd,ch);
    ch:= upcase (ch);
  until (ch in ['Y','N']);
  case ch of
    'Y': mustsave:= true;
    'N': mustsave:= false;
  end;
  textcolor (black);
  writecenter (topline,middle,right,'Overwrite old '+filename+'? (Y/N)');
  textcolor (yellow);
end;

procedure askcompletedeletions;

var ch: char;

begin
  if (maskcount > 0) then begin
    textcolor (lightgray);
    writecenter (topline,middle,right,'Complete deletions? (Y/N)');
    repeat
      read (kbd,ch);
      ch:= upcase (ch);
    until (ch in ['Y','N']);
    case ch of
      'Y': mustsave:= true;
      'N': mustsave:= false;
    end;
  end;
textcolor (black);
writecenter (topline,middle,right,'Complete deletions ? (Y/N)');
textcolor (yellow);
end
else mustsave:= true
end;

procedure saveit;
begin
headnode^.system.name:= ringname;
textcolor (lightgray);
writecenter (topline,middle,right,'Saving');
assign (ringfile,filenname);
rewrite (ringfile);
pointer:= headnode;
repeat
if not (pointer^.deleted)
  then write (ringfile, pointer^.system);
  pointer:= pointer^.fore;
until (pointer= headnode);
close (ringfile);
textcolor (black);
writecenter (topline,middle,right,'Saving');
textcolor (yellow);
end;

procedure FIFO;
var dotpos: integer;
root: string[80];
suffix: string[4];

begin
  dotpos:= pos (',', filenname);
  root:= copy (filenname,1,dotpos-1);
  suffix:= copy (filenname,dotpos,4);
  if (pos (filenname,headnode^.system.name) <> 0)
    then begin
      askcompletedeletions;
      if mustsave
        then begin
          if (suffix= '.SIM')
            then begin
              if fileondisk (root+'.'SIM')
                then begin
                  textcolor (lightgray);
                  writecenter (topline,middle,right,'Saving');

```
assign (ringfile, root+' SIM');
if fileondisk (root+' BAC')
then begin
   rename (ringfile, root+' ZZZ');
   assign (ringfile, root+' BAC');
   rename (ringfile, root+' SIM');
   assign (ringfile, root+' ZZZ');
end;
rename (ringfile, root+' BAC');
saveit;
end
else saveit;
end
else saveit;
end
end
else begin
   if fileondisk (filename)
   then begin
      askoverwrite;
      if mustsave
      then begin
         askcompletedeletions;
         if mustsave
         then saveit;
      end;
   end
else begin
   askcompletedeletions;
   if mustsave
   then saveit;
end;
end;
end;

begin {savering}
   bigwindow (1,1,80,25);
   checknullring;
   if mustsave
   then begin
      ringheader (get_data);
      ringheader (put_data);
     _fifo;
   end;
   mustsave:= false;
end; {savering}

procedure renumber;
const left = 2;
middle = 40;
right = 79;
topline = 2;

var ringnode: node;
doit: boolean;

procedure askrenumber;
var d: datastring;

begin
  textcolor (lightgray);
  writecenter (topline, middle, right, 'Renu1ber? <Y/N> ');
  repeat
    data_control (d, d, wherex, wherey, 0, control);
  until (control in ['Y', 'N']);
  case control of
  'Y': doit := true;
  'N': doit := false;
end;

  textcolor (black);
  writecenter (topline, middle, right, 'Renu1ber? (Y/N)');
  textcolor (yellow);
end;

begin
  bigwindow (1, 1, 80, 25);
  askrenumber;
  if doit then begin
    rubring;
    ringnode := headnode^.back;
    nodenumber := '001';
    repeat
      ringnode^.system.number := nodenumber;
      ringnodes := ringnode^.back;
      nodenumber := incstr (nodenumber);
    until (ringnode = headnode);
    worknode := headnode;
    mustsave := true;
    setring := true;
    setpage := true;
procedure asksaving;
var ch: char;
begin
bigwindow (1,1,80,25);
if mustsave
then begin
writecenter (topline,middle,right,'Save '+filename+' ? (Y/N)');
repeat
read (kbd,ch);
until (upcase (ch) in ['Y','N']);
case ch of
'Y': savering;
'N': ;
end;
end;
end;
end;

procedure thisiring;
const left= 2;
right= 79;
top= 4;
bottom= 21;
var d: datastring;
ringvisible: boolean;
ringnode: node;

\dots

firstline: integer;
topline: integer;
siteline: integer;
centerline: integer;
bottomline: integer;
lastline: integer;
numberpos: integer;
namepos: integer;
datepos: integer;

procedure ringspecs;
begin
  firstline:= 1;
  lastline:= Minlines;
  maskcount:= 0;
  numberpos:= 1;
  namepos:= 9;
  datepos:= 65;
  centerlines:= trunc ((2+lastline-firstline)/2);
  topline:= centerline;
  bottomline:= centerline;
end;

function nodefore (count: integer): node;
var countnode: node;
begin
  countnode:= worknode;
  while (count <> 0)
    do begin
      countnode:= countnodeA.fore;
      count:= count-1;
    end;
  nodefore:= countnode;
end;

function nodeback (count: integer): node;
var countnode: node;
begin
  countnode:= worknode;
  while (count <> 0)
    do begin
      countnode:= countnodeA.back;
      count:= count-1;
    end;
  nodeback:= countnode;
end;

function nodeat (line: integer): node;
var count: integer;
countnode: node;

begin
  countnode:= worknode;
count:= abs (line-siteline);
  while (count <> 0)
    do begin
      if (line > siteline)
        then countnode:= countnode^.back
        else countnode:= countnode^.fore;
      count:= count-1;
    end;
  nodeat:= countnode;
end;

procedure putnode (var ringnode: node; line: integer);

procedure writenode;
var data: datastring;
  n: integer;

begin
  with ringnode^.system
    do begin
      gotoxy (numberpos,line);
      write (number);
      gotoxy (namepos,line);
      write (name);
      gotoxy (datepos,line);
      write (date);
    end;
end;

begin
  with ringnode^ 
    do begin
      if (ringnode= headnode)
        then begin 
          textcolor (cyan);
          writenode;
        end
      else
        if deleted 
          then begin
            textcolor (red);
            writenode;
          end
        else 
          if deleted 
            then begin
              textcolor (red);
              writenode;
            end
    end;
procedure nodeoff (ringnode: node);

begin
  with ringnode^ do begin
    if (ringnode= headnode)
      then begin
        textcolor (cyan);
        writtenode;
      end
    else begin
      textcolor (red);
      writtenode;
    end;
  end;
end;

procedure writtenode;

var data: datastring;
  n: integer;

begin
  with ringnode^ do begin
    if (ringnode= headnode)
      then begin
        textcolor (cyan);
        writtenode;
      end
    else begin
      textcolor (red);
      writtenode;
    end;
  end;
end;
procedure nodeon (ringnode: node);

procedure writenode;

var data: datastring;
n: integer;

begin
  with ringnode^ .system do begin
    gotoxy (numberpos,siteline);
    write (number);
    gotoxy (namepos,siteline);
    write (name);
    gotoxy (datepos,siteline);
    write (date);
  end;
end;

begin
  with ringnode^ do begin
    if (ringnode= headnode) then begin
      textcolor (cyan);
      writenode;
    end
    else
      if deleted then begin
        textcolor (red);
        writenode;
      end
    else
      if (ringnode= worknode) then begin

function dec (line: integer): integer;
begin
  if (line = topline)
  then line := bottomline
  else line := line - 1;
  inc := line;
end;

function inc (var line: integer): integer;
begin
  if (line = bottomline)
  then line := topline
  else line := line + 1;
  inc := line;
end;

procedure rotateup;

procedure makespace;
begin
  bigwindow (winleft, mintop + topline - 1, winright, mintop + bottomline - 1);
  gotoxy (1, 1);
  delline;
  ringwindow;
end;

begin
  ringwindow;
  repeat
    bottomnode := nodeat (bottomline + 1);
    nodeoff (worknode);
    makespace;
    worknode := worknode\^\.back;
    if (bottomnode = worknode)
      then nodeon (bottomnode)
      else begin

putnode (bottomnode,bottomline);
nodeon (worknode);
end;
until (worknode <> headnode);
end;

procedure rotatedown;

procedure makespace;
begin
bigwindow (winleft,wintop+topline-1,winright,wintop+bottomline-1);
gotoxy (1,1);
sinline;
ringwindow;
end;

begin
ringwindow;
repeat
topnode:= nodeat (topline-1);
nodeoff (worknode);
makespace;
worknode:= worknode^.fore;
if (topnode= worknode) then nodeon (topnode)
else begin
putnode (topnode,topline);
nodeon (worknode);
end;
until (worknode <> headnode);
end;

procedure nodeup;
begin
if ringvisible then begin
ringwindow;
repeat
if (siteline= firstline) then rotatedown
else begin
nodeoff (worknode);
worknode:= worknode^.fore;
siteline:= dec (siteline);
nodeon (worknode);
end;
until (worknode <> headnode) or (nodecount= 1);
bigwindow (1,1,80,25);
end
else begin
repeat
  worknode:= worknode^.
fore;
  if (siteline <> firstline)
  then siteline:= dec (siteline);
  until (worknode <> headnode) and not (worknode^.
deleted);
end;
end;

procedure nodedown;
begin
  if ringvisible
  then begin
    ringwindow;
    repeat
      if (siteline= lastline)
      then rotateup
      else begin
        nodeoff (worknode);  
        worknode:= worknode^.
back;
        siteline:= inc (siteline);
        nodeon (worknode);
      end;
    until (worknode <> headnode) or (nodecount= 1);
    bigwindow (1,1,80,25);
  end
  else begin
    repeat
      worknode:= worknode^.
back;
      if (siteline <> lastline)
      then siteline:= inc (siteline);
      until (worknode <> headnode) and not (worknode^.
deleted);
  end;
end;

procedure insertnode;
type datestring= string[10];
var newnode: node;
  line: integer;
procedure makenewnode;

begin
    new (newnode);
    with newnode do begin
        deleted := false;
        system := defaultsystem;
        back := worknode^.back;
        fore := worknode;
    end;
    worknode^.back^.fore := newnode;
    worknode^.back := newnode;
    with newnode^.system do begin
        number := nodenumber;
        name := nodenumber;
        date := currentdate;
    end;
    nodecount := nodecount + 1;
    nodenumber := incstr (nodenumber);
end;

procedure makespace;

begin
    bigwindow (winleft, wintop, winright, wintop + siteline - 1);
    gotoxy (1, 1);
    delline;
    ringwindow;
end;

begin
    ringwindow;
    makenewnode;
    nodeoff (worknode);
    makespace;
    worknodes := newnode;
    nodeon (worknode);
    topline := topline - 1;
    if (topline < firstline) then begin
        if (nodecount <= winlines) then begin
            ringnode := nodefore (siteline - topline);
            putnode (ringnode, topline + nodecount);
            bottomline := bottomline + 1;
            topline := topline + 1;
        end;
    end;
end;
end;
toneline:= firstline;
end;
gotoxy (winxref,siteline);
bigwindow (1,1,80,25);
mustsave:= true;
end; (*insertnode*)

procedure deletenode;
begin
  ringwindow;
  ringnode:= worknode;
  with ringnode do begin
    if deleted then begin
      deleted:= false;
      maskcount:= maskcount-1;
    end
    else begin
      deleted:= true;
      maskcount:= maskcount+1;
    end;
    putnode (ringnode,siteline);
  end;
  bigwindow (1,1,80,25);
mustsave:= true;
end;

procedure pageup;
var line: integer;
begin
  ringwindow;
  clrscr;
  ringnode:= nodeat (topline);
  if (nodecount > winlines) then worknode:= nodefore (winlines-1)
  else worknode:= nodefore (nodecount-1);
  line:= bottomline;
  repeat
    putnode (ringnode,line);
    ringnode:= ringnode^ .fore;
    line:= line-1;
  until (line= topline-1);
  if (worknode= headnode) and (nodecount > 1)
then rotateup;
gotoxy (winxref,siteline);
bigwindow (1,1,80,25);
end;

procedure pagedown;
var line: integer;
begin
  ringwindow;
crerr;
  ringnode:= nodeat (bottomline);
  if (nodecount > winlines)
    then worknode:= nodeback (winlines-1)
    else worknode:= nodeback (nodecount-1);
  line:= topline;
  repeat
    putnode (ringnode,line);
    ringnode:= ringnode^back;
    line:= line+1;
  until (line= bottomline+1);
  if (worknode= headnode) and (nodecount > 1)
    then rotatedown;
gotoxy (winxref,siteline);
bigwindow (1,1,80,25);
end;

procedure putring;
var countup: integer;
  line: integer;
begin
  ringwindow;
  if setring
    then begin
      siteline:= centerline;
      ringnode:= headnode;
      if (nodecount >= winlines)
        then begin
          topline:= firstline;
          bottomline:= lastline;
        end
      else begin
        topline:= centerline-trunc (nodecount/2);
        bottomline:= topline+nodecount-1;
      end;
    end
else begin
  topline:= centerline-trunc (nodecount/2);
  bottomline:= topline+nodecount-1;
end;
ringnode := nodeat (topline);
line := topline;
repeat
  putnode (ringnode,line);
  ringnode := ringnode^.back;
  line := line+1;
until (line = bottomline+1);
if (nodecount > 1)
  then rotateup;
else begin
  ringnode := nodeat (topline);
  line := topline;
  repeat
    putnode (ringnode,line);
    ringnode := ringnode^.back;
    line := line+1;
  until (line = bottomline+1)
end;
setring := false;
ringvisible := true;
end;  (putring)

procedure rubring;
begin
  ringwindow;
  clrscr;
  ringvisible := false;
  bigwindow (1,1,80,25);
end;

procedure thisnode;

procedure nodeheader (dowhat: message);

const left = 2;
xref = 12;
right = 79;
top = 1;
topline = 2;
lines = 2;
bottom = 3;

var line: integer;
procedure itemNodeName (dowhat: message);

const field = 51;
    nameset: charset = ["!'.."~];

begin
    with worknode^system do
        case dowhat of
            put_data: putname (name,xref,line,field);
            get_data: thename (name,xref,line,field,nameset,control);
        end
end;

procedure putitem;

var item: integer;

begin
    with worknode^system do begin
        gotoxy (left+2,topline);
        write (number);
        item:= 0;
        repeat
            line:= topline+item;
            case item of
                0: itemNameode (put_data);
            end;
            item:= item+1;
            until (item>0);
        gotoxy (right-11,topline);
        write (date);
    end;
end;

procedure rubitem;

begin
    bigwindow (left+2,topline,right-2,topline+1);
    gotoxy (1,1);

c:reol;  
bigwindow (1,1,80,25);  
end;  
  
procedure getdata;  
begin  
case control of  
cr: item[page]:= 0;  
endk: item[page]:= 0;  
home: item[page]:= 0;  
end;  
repeat  
lines:= topline+item[page];  
case item[page] of  
0: itemNodename (get_data);  
end;  
case control of  
cr : item[page]:= item[page]+1;  
endk: item[page]:= item[page]-1;  
home: item[page]:= item[page]+1;  
end;  
until not (item[page] in [0])  
    or (control in menuset)  
    or (control in [ccr,pgup,pgdn,uarr,darr,esc]);  
end;  
begin  
bigwindow (1,1,80,25);  
case downwhat of  
put_item: putitem;  
rub_item: rubitem;  
get_data: getdata;  
end  
end; {nodeheader}  

procedure thispage;
procedure pagenuaber (dowhat: message);

const left = 74;
    right = 79;
    top = 3;
    bottom= 5;

procedure putbox;

const y= true;
    n= false;

begin
    textbox (left, top, right, bottom, n, y, y, n, 0, 3, 1, 2);
end;

procedure rubbox;

const y= true;
    n= false;

begin
    textcolor (black);
    putbox;
    textcolor (yellow);
    textbox (left-1, top, right, bottom+1, y, n, y, 0, 0, 0, 0, 0);
end;

procedure putitem;

begin
    gotoxy (left+2, top+1);
    textcolor (cyan);
    write (page:2);
    textcolor (yellow);
end;

procedure rubitem;

begin
    gotoxy (left+2, top+1);
    textcolor (black);
    write (page:2);
    textcolor (yellow);
end;
```pascal
begin
  case dowhat of
    put_box: putbox;
    rub_box: rubbox;
    put_item: putitem;
    rub_item: rubitem;
  end;
end; {pagenumerator}

procedure menufooter (dowhat: message);
const left = 2;
  right = 79;
  top = 22;
  bottom = 25;
  functionmenu = 'Design Simulate Units';
var controlmenu: string[80];

procedure putitem;
begin
  controlmenu :=
    'esc '#+24' '#+25' pgup pgdn '+'
    'ctrl-cr cr '#+26' '#+27' spc';
  gotoxy (whereX, 23);
  center (whereY, left, right, functionmenu);
  writeyellowtips (functionmenu);
  menuset := ['D', 'S', 'U'];
  gotoxy (whereX, 24);
  center (whereY, left, right, controlmenu);
  write (controlmenu);
end;

procedure rubitem;
begin
  bigwindow (left+1, top+1, right-1, bottom-1);
  clrscr;
  bigwindow (1, 1, 80, 25);
end;
```
begin
  case dowhat of
    put_item: putitem;
    rub_item: rubitem;
  end
end;

{$1 page1.inc}$

{$1 page2.inc}$

{$1 page3.inc}$

begin (thispage)
  bigwindow (1,1,80,25);
  textcolor (yellow);
  if setpage then begin
    page:= 1;
    box[1]:= 1;
    box[2]:= 1;
    box[3]:= 1;
    item[1]:= 0;
    item[2]:= 0;
    item[3]:= 0;
    setpage:= false;
  end;
  repeat
    case page of
      1: page1 (box[1],item[1]);
      2: page2 (box[2],item[2]);
      3: page3 (box[3],item[3]);
    end;
  case control of
    pgup: if page= 1 then page:= 3 else page:= page-1;
    pgdn: if page= 3 then page:= 1 else page:= page+1;
  end;
  if control in menuset then begin
    pagonenumber (rub_box);
    pagonenumber (rub_item);
    menufooter (rub_item);
  end;
end;
begin {thisnode}
if not (worknode^*.deleted) and (worknode^>headnode)
then begin
  ringheader (rub_item);
  menufooter (rub_item);
  rubring;
  thispage;
  putring;
  ringheader (put_item);
  menufooter (put_item);
end;
end;

begin {thisring}
ringspecs;
putring;
repeat
  ringwindow;
data_control (d,d,winxref,siteline,0,control);
if scancode
then begin
  case control of
    ins : insertnode;
    del : deletenode;
    uarr : nodeup;
    darr : nodedown;
    pgup : pageup;
    pgdn : pagedown;
  end;
end
else begin
  case control of

cr : thisnode;
'S' : savering;

end;
end;
until (control in ['W', 'R']) and not scancode;
end;

begin (thisfile)
filename := defaultfilename;
mustsave := false;

bigwindow (1,1,80,25);
putframe;
windowspecs;

ringheader (put_item);
menufooter (put_item);
control := 'W';
repeat
    case control of
        'W' : makering;
        'R' : renumber;
    end;
    thisring;
until (control in [esc]);
end;

begin (main)
    preanimator;
    thisfile;
end.
defaultsystem: twotanksystem=

(number: '000';
name : '000';
date : '00/00/0000';

controlmode: PI;
ALPHA: 0.1;
KC: 4;
TI: 1;
TD: 1;

UNITSA: none;
UNITSC: none;
UNITST1: minutes;
UNITST2: minutes;

HT1: 70;
A1: 10;
HT2: 45;
A2: 10;
H3: 0;

UNITSH1: ft;
UNITSA1: sqft;
UNITSH2: ft;
UNITSA2: sqft;
UNITSH3: ft;

H2SB: 20;
THETA1B: 50;
THETA2B: 50;
FLB: 20;

UNITSH2S: ft;
UNITSTH1: percent;
UNITSTH2: percent;
UNITSF1: cfm;

PROFILEH2S: STEP;
PROFILETH1: DESN;
PROFILETH2: DESN;
PROFILE1: DESN;

t0h2s: 0.1;
t1h2s: 0;
SPh2s: 30;
SP1h2s: 0;
SP2h2s: 0;
Wh2s: 0;

UNITSt0h2s: minutes;
UNITSt1h2s: minutes;
UNIT5sph2s: ft;
UNIT5splh2s: ft;
UNIT5sp2h2s: ft;
UNIT5swh2s: cpm;

t0thetal: 0;
t1thetal: 0;
SPthetal: 0;
SP1thetal: 0;
SP2thetal: 0;
Wthetal: 0;

UNITSt0thetal: minutes;
UNITSt1thetal: minutes;
UNIT5spthetal: ft;
UNIT5splthetal: ft;
UNIT5sp2thetal: ft;
UNIT5swthetal: cpm;

t0theta2: 0;
t1theta2: 0;
SPtheta2: 0;
SP1theta2: 0;
SP2theta2: 0;
Wtheta2: 0;

UNITSt0theta2: minutes;
UNITSt1theta2: minutes;
UNIT5sptheta2: ft;
UNIT5spltheta2: ft;
UNIT5sp2theta2: ft;
UNIT5swtheta2: cpm;

t0fl: 0;
t1fl: 0;
SPf1: 0;
SPfl: 0;
SP2f1: 0;
Wf1: 0;

UNITSt0f1: minutes;
UNITSt1f1: minutes;
UNITSpf1: ft;
UNITSpfl: ft;
UNITSp2fl: ft;
UNITSwf1: cpm;

OPCURVEvalve1: LIN;
OPCURVEvalve2: LIN;
OPCURVEtrans2: LIN;
OPCURVEpump: LIN;

CV1: 100;
CV2: 25;

UNITScv1: ft_min;
UNITScv2: ft_min;

Bl: 0;
B2: 0;
BT: 0;
BM: 0;

UNITSb1: none;
UNITSb2: none;
UNITSbt: none;
UNITSbm: none;

H2MIN: 0;
H2MAX: 45;
V2MIN: 0;
V2MAX: 10;

UNITSh2min: ft;
UNITSh2max: ft;
UNITSv2min: v;
UNITSv2max: v;

VMIN: 0;
VMAX: 10;
F1MIN: 0;
F1MAX: 1000;

UNITSvmin: v;
UNITSvmax: v;
UNITSf1min: cfm;
UNITSf1max: cfm;
procedure releaseheap (aheappointer : intpoint);
var i : integer;
begin
i := ((seg(heaptr^) - seg(aheappointer^)) shl 4) +
  (ofs(heaptr^) - ofs(aheappointer^));
free1em(aheappointer,i)
end; {releaseheap}

function integernumber (input: datastring; var variable: integer): boolean;
var code: integer;
hold: integer;
begin
hold:= variable;
val (input,variable,code);
if (code=0) then integernumber:= true
else begin
  integernumber:= false;
  variable:= hold
end
end;

function realnumber (input: datastring; var variable: real): boolean;
var code: integer;
hold: real;

function despace (data: datastring): datastring;
var n: integer;
  puredata: datastring;
begin
  puredata:= '';
  for n:= 1 to length (data)
    do if data[n]<>spc
     then puredata:= puredata+data[n];
  despace:= puredata;
end;
begin
  input := despace (input);
  hold := variable;
  val (input, variable, code);
  if (code = 0) then realnumber := true
  else begin
    realnumber := false;
    variable := hold;
  end
end;

function opcurvesymbol (data: datastring; var curve: scalar): boolean;
begin
  opcurvesymbol := true;
  if (data = 'LIN') then curve := LIN
  else if (data = 'XPO') then curve := XPO
  else opcurvesymbol := false
end;

function modesymbol (data: datastring; var mode: scalar): boolean;
begin
  modesymbol := true;
  if (data = 'P') then mode := P
  else if (data = 'PI') then mode := PI
  else if (data = 'PD') then mode := PD
  else if (data = 'PID') then mode := PID
  else modesymbol := false
end;

function profilesymbol (data: datastring; var profile: scalar): boolean;
begin
  profilesymbol := true;
  if (data = 'DESN') then profile := DESN
  else if (data = 'STEP') then profile := STEP
  else if (data = 'PULS') then profile := PULS
else
if (data = 'RAMP') then profile := RAMP
else
if (data = 'SINE') then profile := SINE
else
profile symbol := false
end;

function unitsymbol (data: datastring; var units: scalar): boolean;
begin
unitsymbol := true;
if (data = 'SEC') then units := sec
else
if (data = 'MIN') then units := minutes
else
if (data = 'HRS') then units := hrs
else
if (data = 'IN') then units := inches
else
if (data = 'FT') then units := ft
else
if (data = 'CM') then units := cm
else
if (data = 'M') then units := m
else
if (data = 'SQIN') then units := sqin
else
if (data = 'SQFT') then units := sqft
else
if (data = 'SQCM') then units := sqcm
else
if (data = 'SQM') then units := sqm
else
if (data = 'CFS') then units := cfs
else
if (data = 'CFM') then units := cfm
else
if (data = 'CFH') then units := cfh
else
if (data = 'LPS') then units := lps
else
if (data = 'LPM') then units := lpm
else
if (data = 'LPH') then units := lph
else
if (data = 'FT^((3/2))/SEC') then units := ft_sec
else
if (data = 'FT^((3/2))/MIN') then units := ft_min
else
  if (data= 'CPS') then units:= cps
else
  if (data= 'CPM') then units:= cpm
else
  if (data= 'CPH') then units:= cph
else
  if (data= 'MV') then units:= mv
else
  if (data= 'V') then units:= v
else
  if (data= 'X') then units:= percent
else
  if (data= '') then units:= none
else
  unitsymbol:= false
end;

procedure putname (name: datastring; xref, line, field: integer);

begin
  gotoxy (xref, line);
  write (name);
end;

procedure putsymbol (symbol: scalar; xref, line, field: integer);

begin
  gotoxy (xref, line);
  write (symbolstring[symbol], spc: (field-length(symbolstring[symbol])));
end;

procedure putmode (mode: scalar; xref, line, field: integer);

begin
  gotoxy (xref, line);
  write (modestring[mode], spc: (field-length(modestring[mode])));
end;

procedure putprofile (profile: scalar; xref, line, field: integer);

begin
  gotoxy (xref, line);
  write (profilestring[profile], spc: (field-length(profilestring[profile])));
end;
procedure putopcurve (opcurve: scalar; xref, line, field: integer);
begin
  gotoxy (xref, line);
  write (opcurvestring[opcurve], spc: (field-length(opcurvestring[opcurve])));
end;

function format (number: real; field: integer; var accur: integer): boolean;
begin
  format := true;
  if (number = 0)
  or (abs(number) < 0.001) and (abs(number) < 1000.0)
  then accur := 3 else
  if (abs(number) >= 1000.0) and (abs(number) < 10000.0)
  then accur := 2 else
  if (abs(number) >= 10000.0) and (abs(number) < 1000000.0)
  then accur := 1 else
  if (abs(number) >= 1000000.0) and (abs(number) < 10000000.0)
  then accur := 0 else
  if (abs(number) >= 0.0001) and (abs(number) < 0.001)
  then accur := 4 else
  if (abs(number) >= 0.00001) and (abs(number) < 0.0001)
  then accur := 5 else
  format := false
end;

function numberstring (number: real; field: integer): datastring;
var temp: datastring;
  accur: integer;
begin
  if format (number, field, accur)
  then str (number, field, accur, temp)
  else str (number, field, temp);
  numberstring := temp;
end;

procedure putnumber (number: real; xref, line, field: integer);
var accur: integer;
begin
  gotoxy (xref, line);
  if format (number, field, accur)
  then write (number, field, accur) else write (number, field);
end;

procedure putunits (units: scalar; xref, line, field: integer);
begin
  gotoxy (xref, line);
  write (unitstring[units], spc: (field-length(unitstring[units])));
end;

procedure putquantity (number: real; units: scalar;
  xref, line, field: integer);
var accur: integer;
begin
  gotoxy (xref, line);
  if format (number, field, accur)
    then write (number: field: accur)
    else write (number: field);

  if (units <> none)
    then write (spc: 3, unitstring[units]);
end;

procedure beep;
begin
  sound (440);
  delay (50);
  nosound;
end;

procedure UNITStocompute;
begin
  with worknode^.system 
  do begin
    ALPHA:= ALPHA / factor[UNITSalpha];
    KC:= KC / factor[UNITSkc];
    TI:= TI / factor[UNITSti];
    TD:= TD / factor[UNITStd];
    HT1:= HT1 / factor[UNITSh1];
    A1:= A1 / factor[UNITSa1];
    HT2:= HT2 / factor[UNITSh2];
    A2:= A2 / factor[UNITSa2];
    H3:= H3 / factor[UNITSh3];
    H2SB:= H2SB / factor[UNITSh2s];
    THETA1B:= THETA1B / factor[UNITSteta1];
THETA2B := THETA2B / factor[UNITSttheta2];
FLB := FLB / factor[UNITSttheta1];

t0h2s := t0h2s / factor[UNITSt0h2s];
t1h2s := t1h2s / factor[UNITSt1h2s];
SPh2s := SPh2s / factor[UNITSpsh2s];
SP1h2s := SP1h2s / factor[UNITSp1h2s];
SP2h2s := SP2h2s / factor[UNITSp2h2s];
Wh2s := Wh2s / factor[UNITSwsh2s];

t0thetal := t0thetal / factor[UNITSt0thetal];
tithetal := tithetal / factor[UNITStithetal];
SPthetal := SPthetal / factor[UNITSpsthetal];
SP1thetal := SP1thetal / factor[UNITSp1thetal];
SP2thetal := SP2thetal / factor[UNITSp2thetal];
Whthetal := Whthetal / factor[UNITSwthetal];

t0theta2 := t0theta2 / factor[UNITSt0theta2];
t1theta2 := t1theta2 / factor[UNITSt1theta2];
SPtheta2 := SPtheta2 / factor[UNITSpstheta2];
SP1theta2 := SP1theta2 / factor[UNITSp1theta2];
SP2theta2 := SP2theta2 / factor[UNITSp2theta2];
Whtheta2 := Whtheta2 / factor[UNITSwtheta2];

t0fl := t0fl / factor[UNITSt0fl];
t1fl := t1fl / factor[UNITSt1fl];
SPfl := SPfl / factor[UNITSpfl];
SP1fl := SP1fl / factor[UNITSp1fl];
SP2fl := SP2fl / factor[UNITSp2fl];
Wfl := Wfl / factor[UNITSwfl];

CV1 := CV1 / factor[UNITScv1];
CV2 := CV2 / factor[UNITScv2];

B1 := B1 / factor[UNITSb1];
B2 := B2 / factor[UNITSb1];
B1 := B1 / factor[UNITSbt];
BM := BM / factor[UNITSbm];

H2MIN := H2MIN / factor[UNITSh2min];
H2MAX := H2MAX / factor[UNITSh2max];
V2MIN := V2MIN / factor[UNITSv2min];
V2MAX := V2MAX / factor[UNITSv2max];

VMIN := VMIN / factor[UNITSvmin];
VMAX := VMAX / factor[UNITSvmax];
F1MIN := F1MIN / factor[UNITSh1min];
F1MAX := F1MAX / factor[UNITSh1max];

end;
end;
procedure UNITStoreturn;

begin
  with worknode^.system do begin
    ALPHA:= ALPHA*factor[UNITSalpha];
    KC:= KC*factor[UNITSkc];
    TI:= TI*factor[UNITSqi];
    TD:= TD*factor[UNITSqd];
    HT1:= HT1*factor[UNITSht1];
    A1:= A1*factor[UNITSa1];
    HT2:= HT2*factor[UNITSht2];
    A2:= A2*factor[UNITSa2];
    H3:= H3*factor[UNITSht3];
    H2SB:= H2SB*factor[UNITSht2s];
    THETA1B:= THETA1B*factor[UNITStheta1];
    THETA2B:= THETA2B*factor[UNITStheta2];
    FLB:= FLB*factor[UNITSfl];
  t0h2s:= t0h2s*factor[UNITSt0h2s];
  t1h2s:= t1h2s*factor[UNITSt1h2s];
  SPh2s:= SPh2s*factor[UNITSp2h2s];
  SP1h2s:= SP1h2s*factor[UNITSp2h2s];
  SP2h2s:= SP2h2s*factor[UNITSp2h2s];
  Wh2s:= Wh2s*factor[UNITSwh2s];
  t0theta1:= t0theta1*factor[UNITSt0theta1];
  t1theta1:= t1theta1*factor[UNITSt1theta1];
  SPtheta1:= SPtheta1*factor[UNITSp2theta1];
  SP1theta1:= SP1theta1*factor[UNITSp2theta1];
  SP2theta1:= SP2theta1*factor[UNITSp2theta1];
  Wtheta1:= Wtheta1*factor[UNITSwtheta1];
  t0theta2:= t0theta2*factor[UNITSt0theta2];
  t1theta2:= t1theta2*factor[UNITSt1theta2];
  SPtheta2:= SPtheta2*factor[UNITSp2theta2];
  SP1theta2:= SP1theta2*factor[UNITSp2theta2];
  SP2theta2:= SP2theta2*factor[UNITSp2theta2];
  Wtheta2:= Wtheta2*factor[UNITSwtheta2];
  t1f1:= t1f1*factor[UNITSt1f1];
  SP1:= SP1*factor[UNITSp1];
  SP1f1:= SP1f1*factor[UNITSp1f1];
  SP2f1:= SP2f1*factor[UNITSp2f1];
  Wf1:= Wf1*factor[UNITSwf1];
CV1 := CV1*factor[UNITScv1];
CV2 := CV2*factor[UNITScv2];

B1 := B1*factor[UNITSbl];
B2 := B2*factor[UNITSbl];
BT := BT*factor[UNITSbt];
BM := BM*factor[UNITSbm];

H2MIN := H2MIN*factor[UNITSh2min];
H2MAX := H2MAX*factor[UNITSh2max];
V2MIN := V2MIN*factor[UNITSv2min];
V2MAX := V2MAX*factor[UNITSv2max];

VMIN := VMIN*factor[UNITSvmin];
VMAX := VMAX*factor[UNITSvmax];
F1MIN := F1MIN*factor[UNITSf1min];
F1MAX := F1MAX*factor[UNITSf1max];
end;
end;

procedure textbox (xlow, ylow, xhigh, yhigh: integer;
top, left, bottom, right: boolean;
T1, T2, T3, T4: integer);

const corner1: char = '?';
corner2: char = 'Z';
corner3: char = '@';
corner4: char = 'Y';
xbar : char = 'D';
ybar : char = '3';
vernex1: char = '4';
vernex4: char = '4';
vernex2: char = 'C';
vernex3: char = 'C';
hornex1: char = 'B';
hornex2: char = 'B';
hornex3: char = 'A';
hornex4: char = 'A';
xbar40 : string[40] = 'DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD';

var x, y: integer;
horline: string[80];

begin
horline := copy (xbar40+xbar40,1,xhigh-xlow-1);
gotoxy (xhigh,ylow);
case T1 of
  1: write (corner1);
2: write (vernex1);
3: write (hornex1);

end;
if top
then begin
  gotoxy (xlow+1,ylow);
  write (horline);
end;
gotoxy (xlow,ylow);
case T2 of
  1: write (corner2);
  2: write (vernex2);
  3: write (hornex2);
end;
if left and right
then begin
  for y:= (ylow+1) to (yhigh-1)
  do begin
    gotoxy (xlow,y);
    write (ybar);
    gotoxy (xhigh,y);
    write (ybar)
  end;
end
else
  if left
  then begin
    for y:= (ylow+1) to (yhigh-1)
    do begin
      gotoxy (xlow,y);
      write (ybar)
    end;
  end
else begin
  for y:= (ylow+1) to (yhigh-1)
  do begin
    gotoxy (xhigh,y);
    write (ybar)
  end;
end;
gotoxy (xlow,yhigh);
case T3 of
  1: write (corner3);
  2: write (vernex3);
  3: write (hornex3);
end;
if bottom
then begin
  gotoxy(xlow+1,yhigh);
  write(horline)
end;
gotoxy(xhigh,yhigh);
case T4 of
  1: write(corner4);
  2: write(vernex4);
  3: write(hornex4);
end
end; {textbox}

procedure putframe;
const y= true;
  n= false;

procedure headbox;
const left = 2;
  right = 79;
  top = 1;
  bottom= 3;
begin
  textbox(left,top,right,bottom,y,y,y,y,1,1,1,1);
end;

procedure pagebox;
const left = 2;
  right = 79;
  top = 3;
  bottom= 22;
begin
  textbox(left,top,right,bottom,n,y,n,y,2,2,0,0);
end;

procedure menubox;
const left = 2;
right = 79;
top = 22;
bottom = 25;

begin
  textbox (left, top, right, bottom, y, y, y, 2, 2, 1, 1);
end;

begin (putframe)
  window (1, 1, B0, 25);
  clrsctr;
  headbox;
  pagebox;
  menubox;
end;

function cap (data: datastring): datastring;
var capdata: datastring;
n: integer;
begin
  capdata := '';
  for n := 1 to length (data)
  do capdata := capdata + upcase (data[n]);
cap := capdata;
end;

procedure center (ypos: integer; x1, x2: integer; data: datastring);
begin
  gotoxy (x1 + trunc (((x2 - x1 + 1) - length (data))/2), ypos);
end;

procedure writecenter (ypos: integer; x1, x2: integer; data: datastring);
begin
  center (ypos, x1, x2, data);
  write (data);
end;

procedure writeyellowtips (data: datastring);
var n: integer;
  datalength: integer;
  ch: char;

begin
  datalength:= length (data);
  n:= 1;
  repeat
    if (data[n] = ' ')
      then repeat
        ch:= data[n];
        n:= n+1;
        write (ch);
      until (data[n] = ' ') or (n=datalength);
    ch:= data[n];
    if (ch = upcase(ch))
      then textcolor (yellow);
      write (ch);
    textcolor (lightgray);
    n:= n+1;
    if (data[n] = ' ')
      then repeat
        ch:= data[n];
        n:= n+1;
        write (ch);
      until (ch = ' ') or (n=datalength);
      until (n=datalength);
  textcolor (yellow);
end;

procedure align (data: datastring; xref, line: integer);
begin
  gotoxy (xref-length (data), line);
  write (data);
end;

procedure header (color: message;
  ypos: integer;
  x1,x2: integer;
  data: datastring);
const leftbloc : char = '\';
rightbloc: char = '^';
bigbloc40: string[40] = '....................................................';

var dataleft: integer;
spcleft: integer;
width: integer;

begin
    width:= x2-x1+1;
    dataleft:= trunc (length (data)/2);
    spcleft:= trunc (width/2)-dataleft-1;
    if (length (data) <= Midth)
    then begin
        gotoxy (x1,ypos);
        case color of
            put_data: textcolor (cyan);
            rub_data: textcolor (black);
        end;
        write (rightbloc);
        write (copy (bigbloc40+bigbloc40,1,spcleft));
        textcolor (black);
        case color of
            put_data: textbackground (cyan);
            rub_data: textbackground (black);
        end;
        write (data);
        textbackground (black);
        case color of
            put_data: textcolor (cyan);
            rub_data: textcolor (black);
        end;
        write (copy (bigbloc40+bigbloc40,1,width-length (data)-spcleft-2));
        write (leftbloc);
        end;
        textcolor (yellow);
    end;

function incstr (numberstr: nodestring): nodestring;

var tempnum: integer;
    code : integer;
    tempstr: nodestring;
    n: integer;

begin
    val (numberstr,tempnum,code);
    tempnum:= tempnum+1;
    str (tempnum:3,tempstr);
    end;
for n:= 1 to 3
  do if tempstr[n] = ' ' 
    then tempstr[n] := '0'; 
    incstr := tempstr; 
  end;

function currentdate: datastring;

  type daterecord =
    record
      ax, bx, cx, dx,
      bp, si, ds, es,
      flags: integer;
    end;

  var datedata: daterecord;
  month: string[2];
  day: string[2];
  year: string[4];
  cx, dx: integer;

  begin
    with datedata do
      ax := #2a shr 8;
      msdos (datedata);
    with datedata do begin
      str(cx, year);
      str(dx mod 256:2, day);
      str(dx shr 8:2, month);
    end;
    if (day[1] = ' ') then day[1] := '0';
    if (month[1] = ' ') then month[1] := '0';
    currentdate := month + '/' + day + '/' + year;
  end;

procedure updatenode;

begin
  with worknode^.system do date := currentdate;
end;

procedure prepanimator;

procedure loadsystem;
var picturefile: file of systeapicture;

begin
  assign (picturefile,'SYSTEM.GRA');
  reset (picturefile);
  read (picturefile,system);
  close (picturefile);
end;

procedure loadvalve;

var picturefile: file of valvepicture;

begin
  assign (picturefile,'VALVE.GRA');
  reset (picturefile);
  read (picturefile,valve);
  close (picturefile);
end;

procedure load02box;

var picturefile: file of boxpicture;

begin
  assign (picturefile,'02BOX.GRA');
  reset (picturefile);
  read (picturefile,02box);
  close (picturefile);
end;

procedure loadwaterh;

var picturefile: file of horslice;

begin
  assign (picturefile,'WATERH.GRA');
  reset (picturefile);
  read (picturefile,waterh);
  close (picturefile);
end;

procedure loadwaterv;

var picturefile: file of verslice;
begin
    assign (picturefile,'WATERV.BRA');
    reset (picturefile);
    read (picturefile, waterv);
    close (picturefile);
end;

begin
    loadsystem;
    loadvalve;
    load02box;
    loadwaterh;
    loadwaterv;
end;
function H2SETPOINT(t: real): real;
var H2S: real;
begin
  with worknodeA.system do case PROFILEh2s of
    DESN: H2SETPOINT:= H2SB;
    STEP: begin
      if t<t0h2s then H2SETPOINT:= H2SB
      else H2SETPOINT:= SPA2h2s;
    end;
    PULS: begin
      if t<t0h2s then H2SETPOINT:= H2SB else
      if t<t1h2s then H2SETPOINT:= SPA2h2s
      else H2SETPOINT:= H2SB;
    end;
    RAMP: begin
      if t<t0h2s then H2SETPOINT:= H2SB else
      if t<t1h2s then H2SETPOINT:= (SPA2h2s-(SPA2h2s-SPAh2s)+(SPA2h2s-SPA2h2s))/(SPA2h2s-t0h2s)*t
      else H2SETPOINT:= SPA2h2s;
    end;
    SINE: begin
      if t<t0h2s then H2S:= H2SB else
      if t<t1h2s then H2S:= H2SB+(SPA2h2s-H2SB)*SIN (SPA2h2s*t)
      else H2S:= H2SB;
      if H2S<0 then H2S:= 0;
      H2SETPOINT:= H2S;
    end;
  end;
end;

function VALVE1(t: real): real;
var THETA1: real;
begin
  with worknodeA.system do case PROFILEthetal of
    DESN: VALVE1:= THETA1B;
    STEP: begin
      if t<t0thetal then VALVE1:= THETA1B
      else VALVE1:= SPA2h2s;
    end;
end;
PULS: begin
  if \( t < t_0 \theta_1 \) then \( \text{VALVE}_1 := \theta_1B \)
  else if \( t < t_1 \theta_1 \) then \( \text{VALVE}_1 := \text{SP}_1 \theta_1 \)
  else \( \text{VALVE}_1 := \theta_1B \)
end;

RAMP: begin
  if \( t < t_0 \theta_1 \) then \( \text{VALVE}_1 := \theta_1B \)
  else if \( t < t_1 \theta_1 \) then \( \text{VALVE}_1 := \text{SP}_1 \theta_1 + (\text{SP}_2 \theta_1 - \text{SP}_1 \theta_1) \frac{(t_1 \theta_1 - t_0 \theta_1)}{t} \)
  else \( \text{VALVE}_1 := \text{SP}_2 \theta_1 \)
end;

SINE: begin
  if \( t < t_0 \theta_1 \) then \( \theta_1 := \theta_1B \)
  else if \( t < t_1 \theta_1 \) then \( \theta_1 := \theta_1B + (\text{SP}_1 \theta_1 - \theta_1B) \sin (\theta_1/t) \)
  else \( \theta_1 := \theta_1B \);
  if \( \theta_1 < 0 \) then \( \theta_1 := 0 \)
  else if \( \theta_1 > 100 \) then \( \theta_1 := 100 \); end;

end;

function \( \text{VALVE}_2(t: \text{real}) : \text{real} \); begin
  \( \text{VAR} \theta_2 : \text{real} \);
  \( \begin{ alignments}
    \text{with worknodeA.system} \\
    \text{do case \text{PROFILE}_2} \text{of}
    \text{DESIGN:} \quad \text{VALVE}_2 := \theta_2B; \\
    \text{STEP:} \begin{ alignments}
      \text{if } t < t_0 \theta_2 \text{ then } \text{VALVE}_2 := \theta_2B \\
      \text{else} \quad \text{VALVE}_2 := \text{SP}_2 \theta_1; \end{alignings} \\
    \text{PULS:} \begin{ alignments}
      \text{if } t < t_0 \theta_2 \text{ then } \text{VALVE}_2 := \theta_2B \text{ else}
      \text{if } t < t_1 \theta_2 \text{ then } \text{VALVE}_2 := \text{SP}_2 \theta_1 \text{ else}
      \text{VALVE}_2 := \theta_2B; \end{alignings} \\
    \text{RAMP:} \begin{ alignments}
      \text{if } t < t_0 \theta_2 \text{ then } \text{VALVE}_2 := \theta_2B \text{ else}
      \text{if } t < t_1 \theta_2 \text{ then } \text{VALVE}_2 := \text{SP}_2 \theta_1 + (\text{SP}_2 \theta_2 - \text{SP}_1 \theta_2) \frac{(t_1 \theta_2 - t_0 \theta_2)}{t} \text{ else}
      \text{VALVE}_2 := \text{SP}_2 \theta_2; \end{alignings} \\
    \text{SINE:} \begin{ alignments}
  \end{alignments} 
\end{alignments}
function \( \text{VALVE}_2(t: \text{real}) : \text{real} \);
if \( t < \theta_2 \) then \( \theta_2 := \theta_2B \) else
if \( t < t_{\theta_2} \) then \( \theta_2 := \theta_2B + (SP_{\theta_2} - \theta_2B) \times \sin (\omega_{\theta_2}t) \)
else \( \theta_2 := \theta_2B \); 
if \( \theta_2 < 0 \) then \( \theta_2 := 0 \) else
if \( \theta_2 > 100 \) then \( \theta_2 := 100 \); 
VALVE2 := \( \theta_2 \); 
end 
end;

function LOAD(t: real): real;
var FL: real;
begin
with worknode\^\textregistered.system
do case PROFILEf1 of
DESN: LOAD := FLB;
STEP: begin
if \( t < t_{\theta 1} \) then \( \text{LOAD} := \text{FLB} \)
else \( \text{LOAD} := \text{SP}_f1 \); 
end;
PULS: begin
if \( t < t_{\theta 1} \) then \( \text{LOAD} := \text{FLB} \) else
if \( t < t_{\theta 1} \) then \( \text{LOAD} := \text{SP}_1 \)
else \( \text{LOAD} := \text{FLB} \); 
end;
RAMP: begin
if \( t < t_{\theta 1} \) then \( \text{LOAD} := \text{FLB} \) else
if \( t < t_{\theta 1} \) then \( \text{LOAD} := \frac{\text{SP}_1f1 + (\text{SP}_2f1 - \text{SP}_1f1)/(t_{\theta 1} - t_{\theta 1})t}{t_{\theta 1} - t_{\theta 1}} \)
else \( \text{LOAD} := \text{SP}_2f1 \)
end;
SINE: begin
if \( t < t_{\theta 1} \) then \( \text{FL} := \text{FLB} \) else
if \( t < t_{\theta 1} \) then \( \text{FL} := \text{FLB} + (\text{SP}_1 - \text{FLB}) \times \sin (\omega_{f1}t) \)
else \( \text{FL} := \text{FLB} \); 
if \( \text{FL} < 0 \) then \( \text{FL} := 0 \); 
\( \text{LOAD} := \text{FL} \); 
end;
end;
end;

function C1(\theta_1: real): real;
begin
with worknode\^\textregistered.system
end;

do case OPCURVEvalve1 of  
  (linear exponential)
  LIN: C1 := CV1*THETA1/100;
  XPO: C1 := CV1*(EXP(B1*THETA1/100)-1)/(EXP(B1)-1);
end
end;

function C2(THETA2: real): real;
begin
  with worknode^.system
  do case OPCURVEvalve2 of
    LIN: C2 := CV2*THETA2/100;
    XPO: C2 := CV2*(EXP(B2*THETA2/100)-1)/(EXP(B2)-1);
  end
end;

function TRANS2(H2: real): real;
begin
  with worknode^.system
  do case OPCURVETrans2 of
    LIN: TRANS2 := V2MIN+(H2-H2MIN)/(H2MAX-H2MIN)*(V2MAX-V2MIN);
    XPO: TRANS2 := V2MIN+(EXP(BT*(H2-H2MIN)/(H2MAX-H2MIN))-1)/
      (EXP(BT)-1)*(V2MAX-V2MIN);
  end
end;

function PUMP(VM: real): real;
begin
  with worknode^.system
  do case OPCURVEpump of
    LIN: PUMP := F1MIN+(VM-VMIN)/(VMAX-VMIN)*(F1MAX-F1MIN);
    XPO: PUMP := F1MIN+(EXP(BM*(VM-VMIN)/(VMAX-VMIN))-1)/(EXP(BM)-1)
      *(F1MAX-F1MIN);
  end
end;
procedure getnumbers;
var K1, K2, K3, K4: real;

procedure calcnextH1;
begin
  with worknode^.system do begin
    K1 := 1/A1*(F1-F2);
    K2 := K1;
    K3 := K1;
    K4 := K1;
    H1 := H1+h/6*(K1+2*K2+2*K3+K4);
  end;
end;

procedure calcnextH2;
var temp: real;
begin
  with worknode^.system do begin
    temp := 1/A2;
    K1 := temp*(F2+LOAD(t)-F0);
    K2 := temp*(F2+LOAD(t+h/2)-F0);
    K3 := temp*(F2+LOAD(t+h/2)-F0);
    K4 := temp*(F2+LOAD(t+h)-F0);
    H2 := H2+h/6*(K1+2*K2+2*K3+K4);
  end;
end;

procedure calcnextX3;
var temp: real;
begin
  with worknode^.system do begin
    if controlmode= PI then begin
      K1 := EV;
      K2 := EV;
      K3 := EV;
      K4 := EV;
      end
    end;
end;
else
if controlmode= PD
then begin
  temp:= 1/ALPHA/TD;
  K1:= temp*(-X3+EV);
  K2:= temp*(-X3+0.5*h*K1+EV);
  K3:= temp*(-X3+0.5*h*K2+EV);
  K4:= temp*(-X3+h*K3+EV);
end
else
if controlmode= PID
then begin
  K1:= X4;
  K2:= X4+0.5*h*K1;
  K3:= X4+0.5*h*K2;
  K4:= X4+h*K3;
end;
X3:= X3+h/6*(K1+2*K2+2*K3+K4);
end;
end;

procedure calcnextX4;
var temp: real;
begi
  with worknodeA.system
  do begin
    if controlmode= PID
    then begin
      temp:= 1/ALPHA/TD;
      K1:= temp*(-X4+EV);
      K2:= temp*(-X4+0.5*h*K1+EV);
      K3:= temp*(-X4+0.5*h*K2+EV);
      K4:= temp*(-X4+h*K3+EV);
      X4:= X4+h/6*(K1+2*K2+2*K3+K4);
    end;
    end;
  end;

procedure updatevectorY;
begi
  with worknodeA.system
  do begin
    H2S:= H2SETPOINT(t);
    THETA1:= VALVE1(t);
    THETA2:= VALVE2(t);
  end;
FL := LOAD(t);
if (H1>H2) then F2 := -C1*THETA1*SQR(H2-H1)
else F2 := C1*THETA1*SQR(H1-H2);
if (H2>H3) then F0 := C2*THETA2*SQR(H2-H3)
else F0 := 0;

V2 := TRANS2(H2);
if V2<0 then V2 := 0;
if V2>10 then V2 := 10;

V2S := TRANS2(H2S);
if V2S<0 then V2S := 0;
if V2S>10 then V2S := 10;

EV := V2S-V2;
EH := H2S-H2;

case controlmode of
  P: VH := VMB+KC*EV;
  PI: VH := VMB+KC/TI*X3+KC*EV;
  PD: VH := VMB+KC/ALPHA*((ALPHA-1)*X3+EV);
  PID: VH := VMB+KC/TI*X3+KC/ALPHA/TI*(SQR(ALPHA)*TD+(1-ALPHA)*TI)*X4
       +KC/ALPHA*EV
end;

if VM<0 then VM := 0;  \{check controller output limits\}
if VM>10 then VM := 10;

F1 := PUMP(VM);
end;
end;  \{updatevectorY\}

begin {getnumbers}
  calcnextH1;
  calcnextH2;
  calcnextX2;
  calcnextX4;
  updatevectorY;
end;

procedure setdesignvector;
begin
with worknode^\$system
  do begin
      H2S:= H2SB;
      THETA1:= THETA1B;
      THETA2:= THETA2B;
      Fl:= FLB;
    H2B:= H2SB;
    H2:= H2B;
    if (H2B>H3) (note, this situation should be prevented at the editor)
      then FOB:= 0
      else FOB:= C2(THETA2B)*SQR(T(H2B-H3));
    FG:= FOB;
    F2B:= FOB-FLB;
    F2:= F2B;
    V2B:= TRANS2(H2B);
    V2:= V2B;
    V2SB:= TRANS2(H2SB);
    V2S:= V2SB;
    EVB:= V2SB-V2B;
    EV:= EVB;
    EH:= H2SB-H2B;
    E:= EH;
    F1:= F2B;
    Fl:= F1B;
    case OPCURVEpump of
      LIN: VMB:= VMIN+(F1B-F1MIN)*(VMAX-VMIN)/(F1MAX-F1MIN);
      XPI: VMB:= VMIN+(VMAX-VMIN)/BM*LN((F1B-F1MIN)/(F1MAX-F1MIN)*(EXP(BM)-1)+1);
    end;
    VM:= VMB;
    H1B:= H2B+SQRT(F2B/C1(THETA1B));
    H1:= H1B;
    X3B:= O;
    X3:= X3B;
    X4B:= O;
    X4:= X4B;
if (HT1 > HT2)
  then F2MAX := C1(100)*SQRT(HT1)
  else F2MAX := C1(100)*SQRT(HT2);

FOMAX := C2(100)*SQRT(HT2-H3);

case PROFILE1 of
  DESN: FLMAX := 0;
  STEP: FLMAX := SPfl;
  PULS: FLMAX := SPf1;
  RAMP: begin
    if (SP1fl > SP2f1)
      then FLMAX := SP1fl
    else FLMAX := SP2f1;
  end;
  SINE: FLMAX := SPf1
end;

FMAX := F1MAX;
if (F2MAX > FMAX)
  then FMAX := F2MAX;
if (FOMAX > FMAX)
  then FMAX := FOMAX;
if (FLMAX > FMAX)
  then FMAX := FLMAX;
end;
end; {setdesignvector}
procedure data_control (olddata : datastring);
  var newdata: datastring;
  xref,line : integer;
  field : integer;
  var control: char
end;

const ctrlset: charset= [cr,spc,bksp];
  scanset: charset= [del,larr,rarr];
  dataset: charset= ['!'..'~'];

var ch: char;
  xbeg,ybeg: integer;
  xend,yend: integer;
  exit: boolean;

procedure getch;
begin
  read (kbd,ch);
  if (ch= esc) and keypressed then begin
    read (kbd,ch);
    scancode:= true;
  end
  else scancode:= false;
end;

procedure backspace;
begin
  if (wherex > xbeg) then gotoxy (wherex-1,wherey);
  exit:= false;
end;

procedure foinspace;
begin
  if (wherex < xbeg+field) then gotoxy (wherex+1,wherey);
  exit:= false;
end;
procedure rub;

var delpos: integer;
    newpos: integer;

begin
    if (wherex > xbeg)
        then begin
            newpos:= wherex-1;
            delpos:= wherex-xbeg;
            delete (newdata,delpos,1);
            gotoxy (newpos,wherey);
            write (copy (newdata,delpos,80));
            write (spc);
            gotoxy (newpos,wherey);
            exit:= false
        end;
end;

procedure deletech;

var delpos: integer;
    newpos: integer;

begin
    newpos:= wherex;
    delpos:= 1+(wherex-xbeg);
    delete (newdata,delpos,1);
    gotoxy (newpos,wherey);
    write (copy (newdata,delpos,80));
    write (spc);
    gotoxy (newpos,wherey);
    exit:= false;
end;

function despace (data: datastring): datastring;

var n: integer;
    puredata: datastring;

begin
    puredata:= '';
    for n:= 1 to length (data)
        do if data[n]<>spc
            then puredata:= puredata+data[n];
    despace:= puredata;
procedure old;

begin
  gotoxy (xbeg,ybeg);
  if (newdata= '')
    then begin
      newdata:= olddata;
      write (olddata);
    end;
  if (newdata<>olddata)
    then mustsave:= true;
  control:= out;
  exit:= true;
end;

procedure new;

begin
  gotoxy (xbeg,ybeg);
  if (despace (newdata)= '')
    then begin
      newdata:= olddata;
      write (olddata);
    end
  else begin
    write (spc:length (newdata));
    newdata:= '';
    gotoxy (xbeg,ybeg);
    end;
  exit:= false;
end;

procedure move;

begin
  if scancode
    then begin
      case ch of
        del : deletech;
        larr: backspace;
        rarr: forspace;
      end;
end
else begin
  case ch of
    cr : old;
    spc : new;
    bksp: rub;
  end;
end;

procedure insertch;
var inspos: integer;
  newpos: integer;
begin
  if (1+where+xbeg > length (newdata)+1)
    then beep
  else if (where + xbeg+field)
    then begin
      gotoxy (where + wherey);
      write (ch);
      newdata= copy (newdata,1,length (newdata)-1)+ch;
      beep
    end
  else if (length (newdata) > field)
    then beep
  else begin
    newpos= where + 1;
    inspos= 1+where+xbeg;
    insert (ch,newdata,inspos);
    gotoxy (newpos-1,wherey);
    write (copy (newdata,inspos,80));
    gotoxy (newpos,wherey);
  end;
  exit:= false;
end;

procedure passcontrol;
begin
  control:= upcase (ch);
  if (newdata= '')
    then newdata= olddata;
  exit:= true;
end;
begin \{(data\_control)\}
xbeg:= xref;
ybeg:= line;
if \{field\} = 0
then xend:= xbeg
else xend:= xbeg+\{newdata\};
yend:= line;
gotoxy \{xend,yend\};
repeat
  getch;
  if \{field\} = 0
    then passcontrol
  else
    if \{ch \in \text{ctrlset}\}
      or scancode and
      \{ch \in \text{scanset}\}
      then move
    else
      if not scancode and
      \{ch \in \text{dataset}\}
      then insertch
      else passcontrol;
  until exit;
end; \{(data\_control)\}
A. ANIMATION - The Graphical Animation

procedure animator;

(* systempicture= array[1..16480] of byte;
valvepicture= array[1..94] of byte;
boxpicture= array[1..138] of byte;
verslice= array[1..9] of byte;
horslice= array[1..73] of byte;*)

(These pictures are stored globally for fast access)

const tmax= 1440.0;  // (24 hours in minutes)
esc = #27;
null= #00;
resratio= 2.2;  // (ratio vertical to horizontal resolution)
maxvtoh= 1.5;  // (max tank height to tank width)
maxwidth= 171;
maxheight= 105;
x1tran= 242;
x2tran= 364;
ytran= 51;
tankbottom= 158;
tank1center= 153;
tank2center= 453;
supplyin= 68;
supplyleft= 28;
supplyright= 34;
pumpbottom= 54;
pumpout= 53;
pipetop= 35;
pipe1bottom= 37;
pipelight= 150;
piperright= 156;
pipelend= 47;
pipe2in= 239;
pipe2out= 367;
pipe2top= 156;
pipe2bottom= 158;
valvelleft= 280;
valveright= 318;
valve2left = 568;
valve2right = 596;
pipeOend = 630;

FOut = 519;
pipeLtop = 13;
pipeLbottom = 15;
valveLright = 502;
valveLleft = 472;
pipeLleft = 450;
pipeLright = 456;
pipeLend = 47;

vwall = 2;
hwall = 1;

(The following is Turbo Graphix Toolbox include file graph.p)

const North = 0;
    East = 90;
    South = 180;
    West = 270;

procedure Graphics;
procedure GraphMode;
procedure GraphColorMode;
procedure HiRes;
procedure HiResColor(Color: Integer);
procedure Palatte(N: Integer);
procedure GraphBackground(Color: Integer);
procedure GraphWindow(X1,Y1,X2,Y2: Integer);
procedure Plot(X,Y,Color: Integer);
procedure Draw(X1,Y1,X2,Y2,Color: Integer);
procedure ColorTable(C1,C2,C3,C4: Integer);
procedure Arc(X,Y,Angle,Radius,Color: Integer);
procedure Circle(X,Y,Radius,Color: Integer);
procedure GetPic(var Buffer; X1,Y1,X2,Y2: Integer);
procedure PutPic(var Buffer; X,Y: Integer);
function GetDotColor(X,Y: Integer): Integer;
procedure FillScreen(Color: Integer);
procedure FillShape(X,Y,FillCol,BorderCol: Integer);
procedure FillPattern(X1,Y1,X2,Y2,Color: Integer);
procedure Pattern(var P);
procedure Back(Dist: Integer);
procedure ClearScreen;
procedure Forwd(Dist: Integer);
function Heading: Integer;
procedure HideTurtle;
procedure Home;
procedure NoWrap;
procedure PenDown;
procedure PenUp;
procedure SetHeading(Angle: Integer);
procedure SetPenColor(Color: Integer);
procedure SetPosition(X,Y: Integer);
procedure ShowTurtle;
procedure TurnLeft(Angle: Integer);
procedure TurnRight(Angle: Integer);
procedure TurtleDelay(Delay: integer);
procedure TurtleWindow(X,Y,W,H: Integer);
function TurtleThere: Boolean;
procedure Wrap;
function Xcor: Integer;
function Ycor: Integer;

{end graph.p}

var ch: char;
dowhat: char;
scancode: boolean;
count: integer;
steps: integer;
backtoeditor: boolean;
newunits: boolean;
continue: boolean;
showdesign: boolean;

(* system: systempicture;
valve: valvepicture;
O2box: boxpicture;
water: horslice;
waterv: verslice;*) (stored globally)
tank1top: integer;
tank2top: integer;
tank1left: integer;
tank2left: integer;
tank1right: integer;
tank2right: integer;
pipeOlevel: integer;
O2 ypos: integer;
PHT1: integer;
PHT2: integer;
PWI: integer;
PW2: integer;

external Graphics[69];
external Graphics[72];
external Graphics[75];
external Graphics[78];
external Graphics[81];
external Graphics[84];
external Graphics[87];
external Graphics[90];
external Graphics[93];
external Graphics[96];
external Graphics[99];
external Graphics[102];
external Graphics[105];
external Graphics[108];
external Graphics[111];
external Graphics[114];
external Graphics[117];
function integernumber (input: datastring; var variable: integer): boolean;

var code: integer;

begin
  val (input, variable, code);
  if (code=0)
    then integernumber := true
    else integernumber := false;
  end;

function realnumber (input: datastring; var parameter: real): boolean;

var code: integer;

begin
  val (input, parameter, code);
  if (code=0)
    then realnumber := true
    else realnumber := false;
  end;

procedure setUNITSruntime;

begin
  with worknode^.system
  do begin
    UNITSh1 := UNITSh1t1;
    UNITSh2 := UNITSh2xs;
    UNITSh3 := none;
    UNITSh4 := none;
    UNITSh5 := UNITSh2xs;
    UNITSt2 := UNITSt2min;
    UNITSt2s := UNITSt2min;
    UNITShv := UNITShvmin;
    UNITSh2 := UNITSh2min;
    UNITSh1 := UNITSh1min; {F2 units same as F1 units}
    UNITSh0 := UNITSh1min; {F0 units same as F1 units}
  end;
end;
procedure loadsystem;
var picturefile: file of systempicture;
begin
  assign (picturefile, 'SYSTEM.GRA');
  reset (picturefile);
  read (picturefile, system);
  close (picturefile);
end;

procedure loadvalve;
var picturefile: file of valvepicture;
begin
  assign (picturefile, 'VALVE.GRA');
  reset (picturefile);
  read (picturefile, valve);
  close (picturefile);
end;

procedure load02box;
var picturefile: file of boxpicture;
begin
  assign (picturefile, '02BOX.GRA');
  reset (picturefile);
  read (picturefile, 02box);
  close (picturefile);
end;

procedure loadwaterh;
var picturefile: file of horslice;
begin
  assign (picturefile, 'WATERH.GRA');
  reset (picturefile);
  read (picturefile, waterh);
  close (picturefile);
end;
procedure loadwaterv;
var picturefile: file of verslice;
begin
  assign (picturefile, 'WATERV.SRA');
  reset (picturefile);
  read (picturefile, waterv);
  close (picturefile);
end;*

procedure drawtanks;

procedure setscales;

const pi = 3.141592654;

var D1, D2: real;
  R1, R2: real;

begin
  with worknode^.system do begin
    D1 := 2*sqrt(A1/pi);
    D2 := 2*sqrt(A2/pi);
    R1 := HT1/D1;
    R2 := HT2/D2;
    if (HT1 >= HT2) then begin
      if (R1 <= maxvtoh) then begin
        if (D1 >= D2) then begin
          PW1 := maxwidth;
          PHT1 := trunc (HT1/D1*PW1/resratio);
          PW2 := trunc (D2/D1*PW1);
          PHT2 := trunc (HT2/HT1*PHT1)
        end
      end else begin
        PW2 := maxwidth;
        PHT1 := trunc ((HT1/D1)*(D1/D2)*D1*maxheight);
        PW1 := trunc (D1/HT1*PHT1*resratio);
        PW2 := trunc (D2/D1*PW1);
        PHT2 := trunc (HT2/HT1*PHT1)
      end
    end else begin
      PW2 := maxwidth;
PHT2 := trunc (HT2/D2*PW2/resratio);
PW1 := trunc (D1/D2*PW2);
PHT1 := trunc (HT1/HT2*PHT2);
end
end
end
else begin
PHT1 := maxheight;
PW1 := trunc (D1/HT1*PHT1/resratio);
PW2 := trunc (D2/D1*PW1);
PHT2 := trunc (HT2/HT1*PHT1);
end
end
else begin
if R2 <= maxvtoh
then begin
if (D2 >= D1)
then begin
PW2 := maxwidth;
PHT2 := trunc (HT2/D2*PW2/resratio);
PW1 := trunc (D1/D2*PW2);
PHT1 := trunc (HT1/HT2*PHT2);
end
else begin
if (R1 <= maxvtoh)
then begin
PHT2 := trunc ((HT2/D2)*(D2/D1)*D2*maxheight);
PW2 := trunc (D2/HT2*PHT2/resratio);
PW1 := trunc (D1/D2*PW2);
PHT1 := trunc (HT1/HT2*PHT2);
end
else begin
PW1 := maxwidth;
PHT1 := trunc (HT1/D1*PW1/resratio);
PW2 := trunc (D2/D1*PW1);
PHT2 := trunc (HT2/HT1*PHT1);
end
end
else begin
PHT2 := maxheight;
PW2 := trunc (D2/HT2*PHT2/resratio);
PW1 := trunc (D1/D2*PW2);
PHT1 := trunc (HT1/HT2*PHT2);
end
end
end
else begin
PHT1 := maxheight;
PW2 := trunc (D2/HT2*PHT2/resratio);
PW1 := trunc (D1/D2*PW2);
PHT1 := trunc (HT1/HT2*PHT2);
end
end
end;
tank1left := tank1center-trunc (PW1/2);
tank1right := tank1center+trunc (PW1/2);
tank1tops := tank1bottom+1-PHT1;
tank2left:= tank2center-trunc (PW2/2);
tank2right:= tank2center+trunc (PW2/2);
tank2top:= tankbottom+1-PHT2;
end;
end;

procedure drawit;

const color= 1;

var xpos,ypos: integer;

begin
  palette (3);
  draw (tank1left-2,tankbottom+1,pipe2in,tankbottom+1,color);
  draw (tank1left-1,tankbottom,tank1left-1,tank1top,color);
  draw (tank1left-2,tankbottom,tank1left-2,tank1top,color);
  draw (tank1right+1,pipe2top-1,tank1right+1,tank1top,color);
  draw (tank1right+2,pipe2top-1,tank1right+2,tank1top,color);
  draw (tank1right+1,pipe2top-1,pipe2in,pipe2top-1,color);
  draw (tank2right+2,tankbottom+1,pipe2out,tankbottom+1,color);
  draw (tank2left-1,pipe2top-1,tank2left-1,tank2top,color);
  draw (tank2left-2,pipe2top-1,tank2left-2,tank2top,color);
  draw (tank2right+1,tankbottom,tank2right+1,tank2top,color);
  draw (tank2right+2,tankbottom,tank2right+2,tank2top,color);
  draw (tank2left-1,pipe2top-1,pipe2out,pipe2top-1,color);
  xpos:= tank1right+vwall+4;
ypos:= pipe2top-hwall;
  repeat
    plot (xpos,ypos,color);
    ypos:= ypos-2;
  until (ypos <= tank1top);
  draw (tank1right+1,tank1top,tank1right+6,tank1top,color);
  draw (tank1right+4,tank1top,tank1right+4,ytran,color);
  draw (tank1right+4,ytran,x1tran,ytran,color);
  xpos:= tank2left-vwall-4;
ypos:= pipe2top-hwall;
  repeat
    plot (xpos,ypos,color);
    ypos:= ypos-2;
  until (ypos <= tank2top);
  draw (tank2left-1,tank2top,tank2left-6,tank2top,color);
  draw (tank2left-4,tank2top,tank2left-4,ytran,color);
  draw (tank2left-4,ytran,x2tran,ytran,color);
end;
begin (drawtanks)
set scales;
draw it;
end;

procedure drawoutvalve;

const cyan= 1;
black= 0;

begin
with worknode-A.system
do begin
    palette(3);
pipeOlevel:= tankbottom-trunc(H3/H72*PHT2)+1;
    if (pipeOlevel < tank2top+2)
then pipeOlevel:= tank2top+2
    else
if (pipeOlevel > tankbottom-1)
then pipeOlevel:= tankbottom-1;

draw (tank2right+1,pipeOlevel-1,tank2right+1,pipeOlevel+1,black);
draw (tank2right+2,pipeOlevel-1,tank2right+2,pipeOlevel+1,black);

draw (tank2right+3,pipeOlevel-2,valve2left,pipeOlevel-2,cyan);
draw (tank2right+3,pipeOlevel+2,valve2left,pipeOlevel+2,cyan);
putpic (valve,valve2left,pipeOlevel+4);
draw (valve2right,pipeOlevel-2,pipe0end,pipe0level-2,cyan);
draw (valve2right,pipeOlevel+2,pipe0end,pipe0level+2,cyan);
end;
end;

procedure draw02box;

const xpos= 523;
    minypos= 48;
    valveclearance= 6;

var ypos: integer;

begin
    ypos:= minypos;
    while ((tank2top-ypos) > valveclearance)
do ypos:= ypos+8;
    if (ypos > minypos)
then ypos:= ypos-8;
putpic (02box,xpos,ypos);
O2ypos:= trunc (ypos)/8);
end;

procedure fillpipes;
const cyan= 1;
begin
  graphwindow (supplyleft,pumpbottom,supplyright,supplyin);
  fillscreen (cyan);
  graphwindow (0,0,639,199);
  draw (pumpout,pipe1top,pumpout-4,pipe1top,cyan);
  draw (pumpout,pipe1top+1,pumpout-3,pipe1top+1,cyan);
  graphwindow (pumpout,pipe1top,pipe1right,pipe1bottom);
  fillscreen (cyan);
  graphwindow (pipe1left,pipe1top,pipe1right,pipe1end);
  fillscreen (cyan);
  graphwindow (tank1right+1,pipe2top,pipe1left,pipe2bottom);
  fillscreen (cyan);
  graphwindow (valve1right,pipe2top,tank2left-1,pipe2bottom);
  fillscreen (cyan);
  graphwindow (tank2right+1,pipe0level-1,pipe1left,pipe2bottom);
  fillscreen (cyan);
  graphwindow (valve2right+1,pipe0level-1,pipe0end,pipe0level+1);
  fillscreen (cyan);
  graphwindow (valveLright+1,pipeLtop,FLout,pipeLbottom);
  fillscreen (cyan);
  graphwindow (pipeLleft,pipeLtop,FLout,pipeLbottom);
  fillscreen (cyan);
  graphwindow (pipeLleft,pipeLtop,FLout,pipeLend);
  fillscreen (cyan);
  graphwindow (0,0,639,199);
end;

procedure writenumber (variable: real);
begin
  if (abs(variable) <= 99.99)
    then write (variable:b:2)
  else
    if (abs(variable) <= 999.9)
      then write (variable:6:1)
else
  if (abs(variable) <= 99999.0)
    then write (variable:6:0)
  else write ('*****');
end;

procedure writetheta (theta: real);
begin
  if (theta= 100)
    then write (theta:4:0)
    else write (theta:4:1);
end;

procedure settime;
const xpos= 35;
ypos= 15;
var hrs,min,sec: integer;
begin
  hrs:= trunc (t/60) mod 24;
  min:= trunc ((t-hrs*60) mod 60;
  sec:= (trunc((t-hrs*60-min)*60)) mod 60;
  gotoxy (xpos,ypos);
  write (hrs:2,':',min:2,':',sec:2);

  (assuming standard time base in min)
end;

procedure setH2s;
const xpos= 37;
ypos= 2;
begin
  with worknode^.system
  do begin
    gotoxy (xpos,ypos);
    if showdesign
      then writelnumber (H2S#factor[UNITSh2s])
    else writelnumber (H2S#factor[UNITSh2s]);
    if newunits
      then begin
          gotoxy (xpos+7,ypos);
        end;
  end;
write (unitstring[UNITSh2s]);
end;
end;
end;

procedure setV1;
const xpos = 37;
ypos = 10;
begin
〈not in use〉
end;

procedure setV2;
const xpos = 36;
ypos = 12;
begin
with worknodeA.system do begin
gotoxy (xpos,ypos);
if showdesign then writenumber (V2B*factor[UNITSh2s])
else writenumber (V2*factor[UNITSh2s]);
if newunits then begin
go toxy (xpos+7,ypos);
write (unitstring[UNITSh2s]);
end;
end;
end;

procedure setVM;
const xpos = 5;
ypos = 2;
begin
with worknodeA.system do begin
gotoxy (xpos,ypos);
if showdesign then writenumber (VMB*factor[UNITSh1m])
else writenumber (VM*factor[UNITSh1m]);
if newunits
then begin
  gotoxy (xpos+7,ypos);
  write (unitstring(UNITSva));
end;
end;
end;

procedure setSUP;

const cyan= 1;
  black= 0;

var y: integer;

begin
  draw (supplyleft,pumpbottom,supplyright,pumpbottom,cyan);
  if (F1<>0)
    then begin
      delay (10+trunc(80*(1-F1/FMAX)));
      draw (supplyleft,pumpbottom,supplyright,pumpbottom,black);
    end
end;

procedure setF1;

const cyan= 1;
  black= 0;
  xpos= 20;
  ypos= 2;

begin
  with worknode^.system
  do begin
    gotoxy (xpos,ypos);
    if showdesign
      then writelnumber (F1B#factor[UNITSf1])
    else writelnumber (F1#factor[UNITSf1]);

    if continue
      then begin
        if (F1<>0)
          then begin
            draw (pipleft,pipelight+1,pipelight,pipelend+1,cyan);
            delay (10+trunc(80*(1-F1/FMAX)));
            draw (pipleft,pipelight+1,pipelight,pipelend+1,black);
          end
        end
  end
end;

if newunits
then begin
  gotoxy (xpos+7,ypos);
  write (unitstring[UNITSf1]);
end;
end;
end;

procedure setH1;

const cyan= 1;
  black= 0;
  xpos= 17;
  ypos= 22;

var surfacenew: integer;
  raster: integer;

procedure checkoverflow;

begin
  if (surfacenew<= tank1top)
    then begin
      surfacenew:= tank1top;
      gotoxy (trunc(tank1left/B)-9,20);
      write ('OVERFLOW');
      end
    else
      if (surfacelold= tank1top) or (H1= H1B)
        then begin
          gotoxy (trunc (tank1left/B)-9,20);
          graphwindow (8*wherey-8,8*wherey-8,8*(wherey+8),8*wherey);
          fillscreen (0);
          graphwindow (0,0,639,199);
          end;
        end;
    end;

begin
  with worknode".system
    do begin
      gotoxy (xpos,ypos);
      if showdesign
        then writelnumber (H1B*factor[UNITSht1])
      else writelnumber (H1*factor[UNITSht1]);
if continue
then begin
  if showdesign
  then surfaceold:= tankbottom;
  raster:= surfaceold;
  surface1new:= tankbottom-trunc(H1/HT1*PHT1);
  checkoverflow;
  if (surface1new <= surface1old)
  then repeat
    draw (tank1left,raster,tank1right,raster,cyan);
    raster:= raster-1;
    until (raster < surface1new)
  else repeat
    draw (tank1left,raster,tank1right,raster,black);
    raster:= raster+1;
    until (raster= surface1new);
  surface1old:= surface1new;
end;

if newunits
then begin
  gotoxy (xpos+7,ypos);
  write (unitstring[UNITSht1]);
end;
end;
end;

procedure setTHETA1;

const xpos= 38;
  ypos= 18;
begin
  with worknode",.system
  do begin
    gotoxy (xpos,ypos);
    if showdesign
    then writetheta (THETA1B*factor[UNITStetal])
    else writetheta (THETA1B*factor[UNITStetal]);

    if newunits
    then begin
      gotoxy (xpos+5,ypos);
      write (unitstring[UNITStetal]);
    end;
  end;
end;
procedure setF2;

const cyan = 1;
    black = 0;
    xpos = 36;
    ypos = 22;

begin
    with worknode^.system
        do begin
            gotoxy (xpos,ypos);
            if showdesign
                then writelnumber (F2B*factor[UNITSf2])
                else writelnumber (F2*factor[UNITSf2]);

            if continue
                then begin
                    if (F2 <> 0)
                        then begin
                            draw (valvelleft,pipe2top, valve1left, pipe2bottom, cyan);
                            draw (valvelright,pipe2top, valve1right, pipe2bottom, cyan);
                            delay (10+trunc(80t(1-F2/FMAX)));
                            draw (valvelleft,pipe2top, valve1left, pipe2bottom, black);
                            draw (valvelright,pipe2top, valve1right, pipe2bottom, black);
                        end;
                    end;

            if newunits
                then begin
                    gotoxy (xpos+7,ypos);
                    write (unitstring[UNITSf2]);
                end;
        end;
end;

procedure setH2;

const cyan = 1;
    black = 0;
    xpos = 55;
    ypos = 22;

var surface2new: integer;
    raster: integer;

procedure checkoverflow;

begin
begin
with worknode^* system
do begin
  gotoxy (xpos,ypos);
  if showdesign
  then writelnumber (H2B*factor[UNITSh2s])
  else writelnumber (H2*factor[UNITSh2s]);
  if continue
  then begin
    if showdesign
    then surface2old:= tankbottom;
    raster:= surface2old;
    surface2new:= tankbottom-trunc(H2/HT2*PH2);
    checkoverflow;
    if (surface2new <= surface2old)
    then repeat
      draw (tank2left,raster,tank2right,raster,cyan);
      raster:= raster-1;
    until (raster < surface2new)
  else repeat
    draw (tank2left,raster,tank2right,raster,black);
    raster:= raster+1;
  until (raster= surface2new);
  surface2old:= surface2new;
end;

if newunits
then begin
  gotoxy (xpos+7,ypos);
  write {unitstring[UNITSh2s]};
end;
end;
procedure setTHETA2;
const xpos= 71;
begin
with worknode^.system
do begin
gotoxy (xpos,02ypos);
if showdesign
then write (THETA2B#factor[UNITSTheta2]:4:1)
else write (THETA2B#factor[UNITSTheta2]:4:1);
if newunits
then begin
  gotoxy (xpos+5,02ypos);
  write (unitstring[UNITSTheta2]);
end;
end;
end;

procedure setFO;
const cyan= 1;
  black= 0;
  xpos= 70;
  ypos= 22;
beg
with worknode^.system
do begin
gotoxy (xpos,ypos);
if showdesign
then writelnumber (FOB#factor[UNITSFo])
else writelnumber (FOB#factor[UNITSFo]);
if continue
then begin
  draw (valve2left-1,pipe0level-1,valve2left-1,pipe0level+1,cyan);
  draw (valve2right+1,pipe0level-1,valve2right+1,pipe0level+1,cyan);
  if (FO <> 0)
  then begin
    draw (pipe0end+1,pipe0level-1,pipe0end+1,pipe0level+1,cyan);
    draw (pipe0end+2,pipe0level-1,pipe0end+2,pipe0level+1,cyan);
    delay (10+trunc (80*(1-FO/FMAX)));
    draw (valve2left-1,pipe0level-1,valve2left-1,pipe0level+1,black);
    draw (valve2right+1,pipe0level-1,valve2right+1,pipe0level+1,black);
  end;
end;
end;
draw (pipeOend+1,pipeOlevel-1,pipeOend+1,pipeOlevel+1,black);
draw (pipeOend+2,pipeOlevel-1,pipeOend+2,pipeOlevel+1,black);
end
end;

if newunits
then begin
  gotoxy (xpos+7,ypos);
  write (unitstring[UNITSfoJ]);
end;
end;
end;

procedure setFL;
const
cyan = 1;
  black = 0;
  xpos = 70;
  ypos = 2;
begin
  with worknode^.system
  do begin
    gotoxy (xpos,ypos);
    if showdesign
    then writelnumber (FLB*factor[UNITSfoJ])
    else writelnumber (FL*factor[UNITSfoJ]);

    if continue
    then begin
      draw (valvelright,pipeltop,valvelright,pipelbottom,cyan);
      draw (valvelleft,pipeltop,valvelleft,pipelbottom,cyan);
      if (FL <> 0)
      then begin
        draw (pipelleft,pipelend+1,pipelright,pipelend+1,cyan);
        delay (10+trunc(80*(1-FL/FMAX)));
        draw (valvelright,pipeltop,valvelright,pipelbottom,black);
        draw (valvelleft,pipeltop,valvelleft,pipelbottom,black);
        draw (pipelleft,pipelend+1,pipelright,pipelend+1,black);
      end
    end;
    if newunits
    then begin
      gotoxy (xpos+7,ypos);
      write (unitstring[UNITSfoJ]);
    end;
  end;
end;
procedure writecenter (ypos: integer; x1,x2: integer; data: datastring);
begin
  gotoxy (x1+trunc (((x2-x1+1)-length (data))/2),ypos);
  write (data);
end;

procedure putmenu;
const menu= 'esc  Current values  Design values  Reset';
begin
  writecenter (25,1,menu);
end;

procedure return;
begin
  continue:= true;
  backtoeditor:= true;
end;

procedure currentvalues;
begin
  if showdesign or continue
    then begin
      continue:= false;
      showdesign:= false;
      setH2S;
      setV2;
      setVM;
      setSUP;
      setF1;
      setH1;
      setTHETA1;
      setF2;
      setH2;
      setTHETA2;
      setFO;
      setFL;
    end
  else
    if not showdesign and not continue
then continue:= true;
end;

procedure designvalues;

begin
if showdesign
then begin
  continue:= true;
  showdesign:= false;
end
else begin
  continue:= false;
  showdesign:= true;
  setH2S;
  setV2;
  setVM;
  setSUP;
  setF1;
  setH1;
  setTHETA1;
  setF2;
  setH2;
  setTHETA2;
  setF0;
  setFL;
end;
end;

procedure reset;

const black= 0;
var raster: integer;

begin
  graphwindow (tank1left,tank1top,tank1right,tankbottom);
  fillscreen (black);
  graphwindow (tank2left,tank2top,tank2right,tankbottom);
  fillscreen (black);
  graphwindow (0,0,639,199);

  setdesignvector;
  showdesign:= true;
  continue:= true;
  setH1;
  setH2;
  showdesign:= false;
newunits:= true;
designvalues;
newunits:= false;
t:= 0;
settime;
end;

procedure checkkbd;
begin
  if keypressed
    then begin
        read (kbd,dowhat);
        if (dowhat= esc) and keypressed
          then begin
              read (kbd,dowhat);
              scancode:= true;
          end
        else scancode:= false;
      end;
end;

begin {animation}
UNITStocompute;
setUNITSruntime;
(*loadsyste1;
loadvalve;
load02box;
loadwaterh;
loadwaterv;*)
hires;
hirescolor (yellow);
putpic (system,0,199);
drawtanks;
drawoutvalve;
draw02box;
fillpipes;
putmenu;

backtoeditor:= false;
reset;
steps:= 0;
repeat
  getnumbers;
  if (steps= 2)
    then begin
steps := 0;
settime;
setH2S;
setV2;
setVM;
setSUP;
setFL;
setH1;
setTHETA1;
setF2;
setH2;
setTHETA2;
setF0;
setFL;
end;

repeat
dowhat := null;
checkkbd;
if not scancode
then case upcase(dowhat) of

esc : return;
'C' : currentvalues;
'D' : designvalues;
'R' : reset;
end;
if (dowhat <> null)
then begin
delay (600);
while keypressed do read (kbd, ch);
end;
until continue;
t := t + h;
steps := steps + 1;
until (t >= tmax) or backtoeditor;
UNITStoreturn;
putframe;
end; {animator}
A. NUMBERS - Columnar Numerical Output

procedure numbers;

  type datastring= string[80];
  headerstr= string[6];
  varstring= string[15];
  charset= set of char;
  message= (put_box,rub_box,put_name,rub_name,
            put_data, rub_data, get_data, put_item, rub_item);

  const left= 3;
  right= 78;
  top= 8;
  bottom= 21;

  headline= 5;
  keymenuine= 24;
  wordmenuine= 23;

  cr = #13;
  lf = #10;
  ack = #06;
  spc = #32;
  bsp= #08;
  ins= #12;
  del = #13;
  larr= #75;
  rarr = #77;
  uarr= #72;
  darr= #80;
  home= #71;
  endk= #79;
  pgup= #73;
  pgdn= #81;
  esc = #27;
  null= #00;

  tmax= 1440;               {24 hours in minutes}
UNITSt: scalar= minutes;  {t is always in minutes}

  field= 7;

  xpos: array[1..8] of integer= (6,15,24,33,42,51,60,69);
  header: array[0..17] of headerstr=
         (' ', 't', 'H1', 'H2', 'X3', 'X4', 'EH',
          'EV', 'V2S', 'V2', 'VM', 'F1', 'F2', 'F0',
          'H2S', 'THETA1', 'THETA2', 'FL');
function integernumber (input: varstring; var variable: integer): boolean;
var code: integer;

function despace (data: datastring): datastring;
var n: integer;
puredata: datastring;

begin
  puredata:= '';
  for n:= 1 to length (data) do if data[n]< ' ' then puredata:= puredata+data[n];
  despace:= puredata;
end;

begin
  input:= despace(input);
  val (input,variable,code);
  if (code=0) then integernumber:= true
else
    integernumber := false;
end;

function realnumber (input: datastring; var variable: real): boolean;
    var code: integer;

    function despace (data: datastring): datastring;
        var n: integer;
            puredata: datastring;

            begin
                puredata := '';
                for n := 1 to length (data)
                    do if data[n] <> spc
                        then puredata := puredata + data[n];
                despace := puredata;
            end;

            begin
                input := despace (input);
                val (input, variable, code);
                if (code = 0) then realnumber := true
            else else realnumber := false;
        end;

        procedure center (ypos: integer; x1, x2: integer; data: datastring);
            begin
                gotoxy (x1 + trunc (((x2 - x1) + 1) + length (data))/2), ypos);
            end;

        procedure writecenter (ypos: integer; x1, x2: integer; data: datastring);
            begin
                center (ypos, x1, x2, data);
                write (data);
            end;

        procedure writeyellowtips (data: datastring);
            var n: integer;
                datalength: integer;
ch: char;

begin
datalength:= length (data);
n:= 1;
repeat
   if (data[n]= ' ')
      then repeat
         ch:= data[n];
         n:= n+1;
         write (ch);
      until (data[n]= ' ') or (n) = datalength;
   ch:= data[n];
   if (ch= upcase(ch))
      then textcolor (yellow);
      write (ch);
      textcolor (lightgray);
      n:= n+1;
   if (data[n]= ' ')
      then repeat
         ch:= data[n];
         n:= n+1;
         write (ch);
      until (ch= ' ') or (n) = datalength;
   until (n) = datalength;
textcolor (yellow);
end;

procedure writenumber (variable: real);
begin
   if (abs(variable) <= 999.999)
      then begin
         write ( variable:7:3);
         if printer
            then write (out,variable:7:3);
      end
   else
      if (abs(variable) <= 9999.99)
         then begin
            write ( variable:7:2);
            if printer
               then write (out,variable:7:2);
         end
      else
         if (abs(variable) <= 99999.9)
            then write (variable:7:1)
         else write ('******');
end;
procedure beep;
begin
  sound(440);
  delay(50);
  nosound;
end;

procedure data_control (olddata: datastring);
  var newdata: datastring;
  field: integer;
  var control: char;
end;

const ctrlset: charset = [cr,spc,bksp];
scanset: charset = [del,larr,rarr];
dataset: charset = ['!',"" , ''];

var ch: char;
  xbeg,ybeg: integer;
  xend,yend: integer;
  exit: boolean;

procedure getch;
begin
  read(kbd,ch);
  if (ch = esc) and keypressed
    then begin
      read(kbd,ch);
      scancode := true;
    end
  else scancode := false;
end;

procedure backspace;
begin
  if (where> > xbeg)
    then gotoxy(where-1,wherey);
  exit := false;
end;

procedure foorspace;
begin
  if (wherex < xbeg+field)
  then gotoxy (wherex+1, wherey);
  exit:= false;
end;

procedure rub;
var delpos: integer;
newpos: integer;

begin
  if (wherex > xbeg)
  then begin
    newpos:= wherex-1;
    delpos:= wherex-xbeg;
    delete (newdata, delpos, 1);
    gotoxy (newpos, wherey);
    write (copy (newdata, delpos, 80));
    write (spc);
    gotoxy (newpos, wherey);
    exit:= false
  end;
end;

procedure deletech;
var delpos: integer;
newpos: integer;

begin
  newpos:= wherex;
  delpos:= 1+(wherex-xbeg);
  delete (newdata, delpos, 1);
  gotoxy (newpos, wherey);
  write (copy (newdata, delpos, 80));
  write (spc);
  gotoxy (newpos, wherey);
  exit:= false;
end;

function despace (data: datastring): datastring;
var n: integer;
puredata: datastring;
begin
puredata := '';
for n := 1 to length (data)
do if data[n] <> spc
then puredata := puredata + data[n];
despace := puredata;
end;

procedure old;
begin
if (newdata = '')
then begin
newdata := olddata;
gotoxy (xbeg, ybeg);
write (olddata);
control := ack;
end
else
if (control = home)
then control := cr
else
if (newdata = olddata)
then control := ack
else control := cr;
exit := true;
end;

procedure new;
begin
gotoxy (xbeg, ybeg);
if (despace (newdata) = '')
then begin
newdata := olddata;
write (olddata);
end
else begin
write (spc: length (newdata));
newdata := '';
gotoxy (xbeg, ybeg);
end;
exit := false;
end;

procedure move;
begin
  if scancode
  then case ch of
    del : deletech;
    larr: backspace;
    rarr: foreshp;
    end
  else case ch of
    cr : old;
    spc: new;
    bksp: rub;
    end;
  end;
end;

procedure insertch;

var inspos: integer;
  newpos: integer;

begin
  if (l+wherex-xbeg > length (newdata)+1)
  then beep
  else if (wherex >= xbeg+field)
  then begin
    gotoxy (wherex-1,wherey);
    write (ch);
    newdata:= copy (newdata,1,length (newdata)-1)+ch;
    beep
  end
  else if (length (newdata) >= field)
  then beep
  else begin
    newpos:= wherex+1;
    inspos:= 1+wherex-xbeg;
    insert (ch,newdata,inspos);
    gotoxy (newpos-1,wherey);
    write (copy (newdata,inspos,80));
    gotoxy (newpos,wherey);
    end;
  exit:= false;
end;

procedure passcontrol;
begin
    control:= upcase (ch);
    exit:= true;
end;

begin {data_control}
    xbeg:= wherex;
    ybeg:= wherey;
    xend:= wherex+length (newdata);
    yend:= wherey;
    gotoxy (xend,yend);
    repeat
        getch;
        if (field= 0)
            then passcontrol
        else
            if (ch in ctrlset)
                or scancode and
                (ch in scanset)
            then move
            else
                if not scancode and
                (ch in dataset)
            then insertch
                else passcontrol;
        until exit;
end; {data_control}

procedure rubwindow;
begin
    bigwindow (4,4,77,21);
    clrscr;
    bigwindow (1,1,80,25);
end;

procedure align (data: datastring; xref,line: integer);
begin
    gotoxy (xref-length (data),line);
    write (data);
end;

procedure putinteger (number: integer; xref,line,field: integer);
begin
    gotoxy (xref,line);
write (number:field);
end;

procedure getinteger (var variable: integer;
  xref,line: integer;
  field: integer;
  min,max: integer;
  enforce: boolean;
  var control: char );
var olddata: datastring;
  newdata: datastring;
  correct: boolean;
  limit: integer;

procedure checknumber;
begin
  gotoxy (xref,line);
  if integernumber (newdata,variable)
  then begin
    if (variable)= min and (variable<= max)
      then correct:= true
    else
      if enforce or not enforce and (variable < min)
        then correct:= false
      else begin
        correct:= true;
        beep;
        end;
  end
  else correct:= false;
  if correct
    then write (variable:field)
  else beep;
end;

procedure showlimits;
begin
  textcolor (cyan);
  gotoxy (xref,line);
  case limit of
    1: str (min:field,newdata);
    2: str (max:field,newdata);
end;
write (newdata);
gotoxy (xref,line);
limit:= limit mod 2 +1;
textcolor (yellow);
end;

begin (getinteger)
str (variable:field,olddata);
newdata:= olddata;
limit:=1;
gotoxy (xref,line);
repeat
data_control (olddata,newdata,field,control);
case control of
  cr : checknumber;
  home: showlimits;
end;
until (control= cr) and correct or not (control in [cr,home]);
end;

procedure keymenu (dowhat: message);
var menu: datastring;

procedure putitem;

begin
  menu:= 'esc '+'26' '+'27' '+'24' '25' 'cr spc';
  writecenter (keymenuline,left,right,menu);
end;

procedure rubitem;

begin
  bigwindow (left,keymenuline,right,keymenuline+1);
gotoxy (1,1);
clear;
bigwindow (1,1,80,25);
end;

begin
case dowhat of
    put_item: putitem;
    rub_item: rubitem;
end;
end;

procedure wordmenu (dowhat: message);
const menu=
'Current values  Design values  Reset  Interval  Scroll  Hardcopy';

procedure putitem;
begin
    center (wordmenu, left, right, menu);
    writeyellowtips (menu);
end;

procedure rubitem;
begin
    bigwindow (left, keymenu, right, wordmenu);
    gotoxy (2, I);
    clrsetp;
    bigwindow (1, 1, 80, 25);
end;

begin
    case dowhat of
        put_item: putitem;
        rub_item: rubitem;
    end;
end;

procedure setmenuitem (itemstr: datastring; on: boolean);
var menu: datastring;
    xstring: integer;
    xscreen: integer;
    x0screen: integer;
begin
  menu:='Current values Design values '+
    'Reset Interval Scroll Hardcopy';
x0screen:=left+trunc(((right-left+l)-length(menu))/2);
xstring:=pos(itemstr,menu);
xscreen:=x0screen+xstring-1;
gotoxy(xscreen,wordmenuine);
if on
  then write(itemstr)
  else writeyellowtips(itemstr);
end;

procedure setUNITSruntime;
begin
  with worknode^.system
  do begin
    UNITSh1:=UNITSht1;
    UNITSh2:=UNITSht2;
    UNITSh3:=none;
    UNITSh4:=none;
    UNITSh:=UNITSh2;
    UNITSh:=UNITSh2;
    UNITSh:=UNITSh;
    UNITSh:=UNITSh;
    UNITSv:=UNITSh;
    UNITSv:=UNITSh;
    UNITSv:=UNITSh;
    UNITSv:=UNITSh;
    UNITSv:=UNITSh;
    (F2 units same as F1 units)
    UNITSv:=UNITSh;
    (F0 units same as F1 units)
  end;
end;

procedure loadunits;
var count: integer;

begin
  for count:=0 to 17
  do with worknode^.system
  do case count of
    0: UNITS[count]:=none;
    1: UNITS[count]:=UNITSh;
    2: UNITS[count]:=UNITSh1;
    3: UNITS[count]:=UNITSh2;
    4: UNITS[count]:=UNITSh3;
    5: UNITS[count]:=UNITSh4;
    6: UNITS[count]:=UNITSh
procedure return;
begin
    continue:= true;
    backtoeditor:= true;
end;

procedure nextcolumn;
begin
    case dowhat of
    larr: if column=1 then column:= 8 else column:= column-1;
    rarr: column:= column mod 8 +1;
    end;
end;

procedure showvariables;

function variableok: boolean;
begin
    with worknode^system
    do begin
        if (header[index[column]]= 'X3') and (controlmode= P)
        then variableok:= false
        else
            if (header[index[column]]= 'X4') and (controlmode in [P,P1])
            then variableok:= false
            else variableok:= true;
    end;
end;
begin
  gotoxy (xpos[column],headline);
  repeat
    case dowhat of
      uarr: if index[column] = 17
        then index[column] := 0
        else index[column] := index[column] + 1;
      darr: if index[column] = 0
        then index[column] := 17
        else index[column] := index[column] - 1;
    end;
  until variableok;
  write (header[index[column]],spc:(field-length(header[index[column]])));
  textcolor (lightgray);
  gotoxy (xpos[column]+2,headline+1);
  write (unitstring[UNITS[index[column]]],
    spc:(field-length(header[index[column]])));
  textcolor (yellow);
  newheaders := true;
end;

procedure writeblanks (num: integer);
begin
  write (spc:num);
  if printer
    then write (out,spc:num);
end;

procedure writecurrentline;
begin
  with worknode^.system
  do for col:= 1 to B
  do begin
    case index[col] of
      0 : writeblanks (7);
      1 : writenumber (t);
      2 : writenumber (H1*factor[UNITSh1]);
      3 : writenumber (H2*factor[UNITSh2]);
      4 : writenumber (X3*factor[UNITSh3]);
procedure writedesignline;
begin
  with worknode^.system do for col:= 1 to 8 do begin
    case index[col] of
      0: writeblanks (7);
      1: writelnumber (t);
      2: writelnumber (H1B*factor[UNITSh1]);
      3: writelnumber (H2B*factor[UNITSh2]);
      4: writelnumber (X3B*factor[UNITSh3]);
      5: writelnumber (X4B*factor[UNITSh4]);
      6: writelnumber (EHB*factor[UNITSeh]);
      7: writelnumber (EVB*factor[UNITSev]);
      8: writelnumber (V2SB*factor[UNITSv2s]);
      9: writelnumber (V2B*factor[UNITSv2]);
     10: writelnumber (VMB*factor[UNITSvm]);
     11: writelnumber (FB1B*factor[UNITShf1]);
     12: writelnumber (FB2B*factor[UNITShf2]);
     13: writelnumber (F0B*factor[UNITSfo]);
     14: writelnumber (H2SB*factor[UNITSh2s]);
     15: writelnumber (THETA1B*factor[UNITStheta1]);
     16: writelnumber (THETA2B*factor[UNITStheta2]);
     17: writelnumber (FLB*factor[UNITSfl]);
    end;
    writeblanks (2); end; end;
end;

procedure currentvalues;

begin
  if showdesign or continue
  then begin
    setmenuItem ('Current values',true);
    setmenuItem ('Design values',false);
    continue:= false;
    showdesign:= false;
    gotoxy (xpos[1],ypos);
    if printer
    then begin
      writeln (out);
      write (out,spc:5);
    end;
    writecurrentline;
    if printer
    then begin
      writeln (out);
      writeln (out);
    end;
  end
  else
    if not showdesign and not continue
    then begin
      setmenuItem ('Current values',false);
      continue:= true;
    end;
end;

procedure designvalues;

begin
  if showdesign
  then begin
    setmenuItem ('Design values',false);
    continue:= true;
    showdesign:= false;
  end
  else begin
    setmenuItem ('Design values',true);
    setmenuItem ('Current values',false);
    continue:= false;
    showdesign:= true;
    gotoxy (xpos[1],ypos);
    if printer
then begin
  write (out, cr);
  write (out, lf);
  write (out, spc: 5);
end;
textcolor (cyan);
write (designline);
textcolor (yellow);
if printer then begin
  writeln (out);
  writeln (out);
end;
end;

procedure reset;
begin
  setmenuitem ('Reset', true);
  t := 0;
  setdesignvector;
  setmenuitem ('Reset', false);
  showdesign := false;
  designvalues;
end;

procedure interval;
const xref = 40;
  line = keymenuline;
var control: char;

procedure itemSTEPS {dowhat: message};
const field = 4;
  accur = 0;
  min = 1;
  max = 9999;
  enforce = true;
begin
  case dowhat of
  put_name: align ('STEPS= ', xref, line);
put_data: putinteger (STEPS, xref, line, field);
get_data: getinteger (STEPS, xref, line, field, min, max, enforce, control);

end
end;

procedure rubitem;
begin
    bigwindow (left+1, keymenulinie, right-1, keymenulinie+1);
gotoxy (1, 1);
cleol;
bigwindow (1, 1, 80, 25);
end;

begin
    setmenuitem ('Interval', true);
    setmenuitem ('Current values', false);
    setmenuitem ('Design values', false);
    keymenu (rub_item);
    itemSTEPS (put_name);
    itemSTEPS (put_data);
    itemSTEPS (get_data);
delay (750);
rubitem;
    setmenuitem ('Interval', false);
    if showdesign
        then setmenuitem ('Design values', true)
    else setmenuitem ('Current values', true);

    keymenu (put_item);
    continue:= false;
end;

procedure defaultheaders;
begin
    index[1]:= 1;
    index[2]:= 2;
    index[3]:= 3;
    index[4]:= 14;
    index[5]:= 10;
    index[6]:= 11;
procedure putheaders;

begin
  if printer then begin
    writeln (out);
    writeln (out);
    write (out,spc:5);
    for col:= 1 to 8 do begin
      write (out,header[index[col]]);
      write (out,spc:(field-length (header[index[col]])+2));
    end;
    writeln (out);
    write (out,spc:5);
    for col:= 1 to 8 do begin
      write (out,'',unitstring[UNITS[index[col]]]);
      write (out,spc:(field-length (unitstring[UNITS[index[col]]])));
    end;
    writeln (out);
    writeln (out);
  end
  else begin
    for col:= 1 to 8 do begin
      gotoxy (xpos[col],headline);
      write (header[index[col]]);
      gotoxy (xpos[col]+2,headline+1);
      textcolor (lightgray);
      write (unitstring[UNITS[index[col]]]);
      textcolor (yellow);
    end;
    end;
  newheaders:= false;
end;

procedure checkkbd;

begin
  if keypressed then begin
    read (kbd,dowhat);
    if (dowhat= esc) and keypressed
then begin
  read (kbd,dowhat);
  scancode := true;
  end
else scancode := false;
dowhat := upcase(dowhat);
end
end;

procedure scroll;

begin
  if scrolling
    then begin
      setmenuitem('Scroll',false);
      scrolling := false;
      bigwindow(left,top,right,bottom);
      clrscr;
      bigwindow(1,1,80,25);
ypos := top;
if showdesign
  then begin
designvalues;
  designvalues;
  end
else
  if not continue
    then begin
  currentvalues;
  currentvalues;
  end;
  end
else begin
  setmenuitem('Scroll',true);
  scrolling := true;
end;
end;

procedure rotatedown;

begin
  bigwindow(left,top,right,bottom);
gotoxy(1,1);
sinline;
bigwindow(1,1,80,25);
end;
procedure hardcopy;

begin
  if printer
    then begin
      setmenuitem ('Hardcopy',false);
      assign (out,'con:');
      printer:= false;
    end
  else begin
    setmenuitem ('Hardcopy',true);
    assign (out,'lst:');
    printer:= true;
    newheaders:= true;
  end;
  if continue
    then if showdown
        then setmenuitem ('Design values',true)
        else setmenuitem ('Current values',true);
  continue:= false;
end;

begin (numbers)
setUNITSruntime;
loadunits;
UNITStocompute;
setdesignvector;
keymenu (put_item);
wordmenu (put_item);

backtoeditor:= false;
newheaders:= true;
showdesign:= false;
continue:= true;
scrolling:= false;
printer:= false;
dowhat:= null;
defaultheaders;
putheaders;
column:= 1;
STEPS:= 2;

t:= 0;
count:= 0;
ypos:= top;
designvalues;
repeat
  getnumbers;
if (count = STEPS)
then begin
  count := 0;
repeat
  dowhat := null;
  delay (50);
  checkkbd;
  if (dowhat in [esc, 'C', 'D', 'R', 'I', 'S', 'H']) and not scancode
  then begin
    if continue
    then wait := 300
    else wait := 0;
    case dowhat of
      esc : return;
      'C' : currentvalues;
      'D' : designvalues;
      'R' : reset;
      'I' : interval;
      'S' : scroll;
      'H' : hardcopy;
    end;
  delay (wait);
  end;
if not continue
then begin
  gotoxy (xpos[column] + length(header[index[column]]), headline);
  if scancode
  then case dowhat of
    larr: nextcolumn;
    rarr: nextcolumn;
    uarr: showvariables;
    darr: showvariables;
  end
else case dowhat of
  cr: nextcolumn;
end;
end; until continue;
if not backtoeditor
then begin
  if newheaders
  then begin
    putheaders;
  end;
if scrolling
then begin
  if ypos = bottom
  then begin
    bigwindow (left, top, right, bottom);
    clrscl;
    bigwindow (1, 1, 80, 25);
    ypos := top;
  end
  else ypos := ypos + 1;
end;
gotoxy (xpos[1], ypos);
if printer
then begin
  write (out, cr);
  write (out, spc: 5);
end;
writecurrentline;
if printer
then begin
  write (out, lf);
end;
t := t + h;
count := count + 1;
until backtoeditor or (t >= tmax);
UNITStoreturn;
rubwindow;
end; {numbers}
procedure page1 (var box, item: integer);
var line: integer;

procedure controls (dowhat: message);
const left = 24;
xref = 40;
right = 56;

top = 5;
topline = 7;
lines = 5;
bottom = 12;

var doALPHA: boolean;
doKC : boolean;
doTI : boolean;
doTD : boolean;

DISP: array[0..4] of integer;

procedure newitem;

procedure SETp;
begin
doALPHA := false;
doKC := true;
doTI := false;
doTD := false;
end;

procedure SETpi;
begin
doALPHA := false;
doKC := true;
doTI := true;
doTD := false;
end;
procedure SETpd;
begin
  doALPHA := true;
  doKC := true;
  doTI := false;
  doTD := true;
end;

procedure SETpid;
begin
  doALPHA := true;
  doKC := true;
  doTI := true;
  doTD := true;
end;

procedure DISPp;
begin
  DISP[0] := 0;
  DISP[1] := 1;
end;

procedure DISPpi;
begin
  DISP[0] := 0;
  DISP[1] := 1;
  DISP[2] := 2;
end;

procedure DISPpd;
begin
  DISP[0] := 0;
  DISP[1] := 1;
  DISP[2] := 2;
  DISP[3] := 3;
end;

procedure DISPpid;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
  DISP[4]:= 4;
end;

begin
with worknode^system do begin
  case controlmode of
    P : SETp;
    PI : SETpi;
    PD : SETpd;
    PID: SETpid;
  end;
  case controlmode of
    P : DISPp;
    PI : DISPpi;
    PD : DISPpd;
    PID: DISPpid;
  end;
end;
end;

procedure itemControlmode (dowhat: message);

const field= 3;
  modeset: scalarset= [P,PI,PD,PID];
begin
  with worknode^system do
    case dowhat of
      put_name: align ('Control mode= ',xref,line);
      put_data: putsymbol (controlmode,xref,line,field);
      get_data: thesymbol (controlmode,xref,line,field,
      modeset,control);
    end;
  end;
end;
procedure itemALPHA (dowhat: message);

const field= 8;
min= 0;
max= 1.0;
nullset: scalarset= [none];
enforce= true;

begin
with worknode^.system do

begin
with worknode^.system do

end;
end;

procedure itemKC (dowhat: message);

const field= 8;
min= 0;
max= 999.990;
nullset: scalarset= [none];
enforce= true;

begin

begin

end;
begin
with worknode^ . system do
  case dowhat of
    put_name: align ('TI=', xref, line);
    put_data: putquantity (TI, UNITSti, xref, line, field);
    get_data: thequantity (TI, UNITSti, xref, line, field,
      min, max, timeset, enforce, control);
  end
end;

procedure itemTD (dowhat: message);
const field = 8;
  min = 0;
  max = 999.999;
  timeset: scalarset = [sec, minutes, hrs];
  enforce = true;
begin
  with worknode^ . system do
    case dowhat of
      put_name: align ('TD=', xref, line);
      put_data: putquantity (TD, UNITStd, xref, line, field);
      get_data: thequantity (TD, UNITStd, xref, line, field,
        min, max, timeset, enforce, control);
    end
  end;
end;

procedure putbox;
const y = true;
  n = false;
var item: integer;
begin
  header (put_data, top, left, right, 'Controls');
  textbox (left, top, right, bottom, n, y, y, 0, 0, 1, 1);
  line:= topline;
  itemControlmode (put_name);
end;
procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;

var item: integer;

begin
  for item := 0 to 4
    do begin
      line := topline + DISP[item];
      case item of
        0: itemControlmode (put_data);
        1: if doALPHA
          then begin
              itemALPHA (put_name);
              itemALPHA (put_data);
            end;
        2: if doKC
          then begin
              itemKC (put_name);
              itemKC (put_data);
            end;
        3: if doTI
          then begin
              itemTI (put_name);
              itemTI (put_data);
            end;
        4: if doTD
          then begin
              itemTD (put_name);
              itemTD (put_data);
            end;
      end;
    end;
end;

procedure rubitem;

begin
  window (kref, topline, right-1, topline+1);
  gotoxy (1, 1);
procedure getdata;

procedure getControlmode;

var oldmode: scalar;

begin
with worknode\system do begin
oldmode:= controlmode;
itemControlmode (get_data);
if (controlmode<>oldmode)
then begin
rubitem;
newitem;
putitem;
end;
end;
end;

begin
control of

cr : item:= 0;
home: item:= 4;
endk: item:= 0;
ccr : item:= 0;
warr: item:= 0;
darr: item:= 0;
end;
repeat
line:= topline+DISP(item);
  case item of
    0: getControlmode;
    1: if doALPHA then itemALPHA (get_data);
    2: if doKC then itemKC (get_data);
    3: if doTI then itemTI (get_data);
    4: if doTD then itemTD (get_data);
end;
  case control of
    cr : item:= item+1;
    home: item:= item-1;
    endk: item:= item+1;
end;
  until not (item in [0..4])
    or (control in menuset)
    or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (controls)
  newitem;
  case dowhat of
    put_box: putbox;
    rub_box: rubbox;
    put_item: putitem;
    rub_item: rubitem;
    get_data: getdata;
  end
end;

procedure physicalspecs (dowhat: message);

const left= 10;
  xref= 18;
  right= 37;
  top= 14;
  topline= 16;
  lines= 5;
  bottom= 21;

var line: integer;

procedure itemHT1 (dowhat: message);

const field= 8;
  min= 0;
  max= 999,990;
  lengthset: scalarset= [inches,ft,cm,m];
  enforce= true;
begin
    with worknode^.system do
    case dowhat of
    put_name: align ('HT1= ',xref,line);
    put_data: putquantity (HT1,UNITSh1,xref,line,field);
    get_data: thequantity (HT1,UNITSh1,xref,line,field,
                           min,max,lengthset,enforce,control);
    end
end;

procedure itemAl (dowhat: message);
const field= 8;
    min= 0.001;
    max= 999.990;
    areaset: scalarset= [sqin,sqft,sqcm,sqmi];
    enforce= true;
begin
    with worknode^.system do
    case dowhat of
    put_name: align ('Al= ',xref,line);
    put_data: putquantity (Al,UNITsal,xref,line,field);
    get_data: thequantity (Al,UNITsal,xref,line,field,
                           min,max,areaset,enforce,control);
    end
end;

procedure itemHT2 (dowhat: message);
const field= 8;
    min= 0;
    max= 999.990;
    lengthset: scalarset= [inches,ft,cm,m];
    enforce= true;
begin
    with worknode^.system do
    case dowhat of
    put_name: align ('HT2= ',xref,line);
    put_data: putquantity (HT2,UNITSh2,xref,line,field);
    get_data: thequantity (HT2,UNITSh2,xref,line,field,
                           min,max,lengthset,enforce,control);
procedure itemA2 (dowhat: message);

const field= 8;
  min= 0.001;
  max= 999.990;
  areaset: scalarset= [sqin,sqft,sqcm,sqm];
  enforce= true;

begin
  with worknode'.system do
    case dowhat of
      put_name: align ('A2= ',xref,line);
      put_data: putquantity (A2,UNITSa2,xref,line,field);
      get_data: thequantity (A2,UNITSa2,xref,line,field,
                               min,max,areaset,enforce,control);
    end
  end
end;

procedure itemH3 (dowhat: message);

const field= 8;
  min= 0;
  (max= H2SB)
  lengthset: scalarset= [inches,ft,cm,m];
  enforce= true;

begin
  with worknode'.system do
    case dowhat of
      put_name: align ('H3= ',xref,line);
      put_data: putquantity (H3,UNITSh3,xref,line,field);
      get_data: thequantity (H3,UNITSh3,xref,line,field,
                               min,H2SB/factor[UNITSh2s],
                               lengthset,enforce,control);
    end
  end
end;

procedure putbox;
const y = true;
    n = false;

var item: integer;

begin
    header (put_data,top,left,right,'Physical specs');
    textbox (left,top,right,bottom,n,y,y,y,0,0,1,1);
    for item:= 0 to 4
        do begin
            line:= topline+item;
            case item of
                0: itemHT1 (put_name);
                1: itemA1 (put_name);
                2: itemHT2 (put_name);
                3: itemA2 (put_name);
                4: itemH3 (put_name);
            end
        end;
    end;

procedure rubbox;

begin
    window (left,top,right,bottom);
    clrsr;
    window (1,1,80,25);
end;

procedure putdata;

var item: integer;

begin
    for item:= 0 to 4
        do begin
            line:= topline+item;
            case item of
                0: itemHT1 (put_data);
                1: itemA1 (put_data);
                2: itemHT2 (put_data);
                3: itemA2 (put_data);
                4: itemH3 (put_data);
            end
procedure rubdata;
begin
  window (xref,topline,right-1,bottom-1);
  clrscr;
  window (1,1,80,25);
end;

procedure getdata;
begin
  case control of
    cr : item:= 0;
    home: item:= 4;
    endk: item:= 0;
    ccr : item:= 0;
  end;
  repeat
    line:= topline+item;
    case item of
      0: itemHT1 (get_data);
      1: itemA1 (get_data);
      2: itemHT2 (get_data);
      3: itemA2 (get_data);
      4: itemH3 (get_data);
    end;
    case control of
      cr : item:= item+1;
      home: item:= item-1;
      endk: item:= item+1;
    end;
    until not (item in [0..4])
      or (control in menuset)
      or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
  end;
begin
case dowhat of
  put_box : putbox;
  rub_box : rubbox;
  put_data: putdata;
  rub_data: rubdata;
  get_data: getdata;
end
end; {physicalspecs}

procedure designconditions (dowhat: message);
const left= 43;
  xref= 54;
  right= 70;
  top= 14;
  topline= 16;
  lines= 4;
  bottom= 20;
var line: integer;

procedure itemH2SB (dowhat: message);
const field= 8;
{min= H3}
{max= HT2}
  lengthset: scalarset= [inches, ft, cm];
  enforce= true;
begin
  with worknode\system do
    case dowhat of
    put_name: align ('H2s=', xref, line);
    put_data: putquantity (H2SB, UNITSh2s, xref, line, field);
    get_data: thequantity (H2SB, UNITSh2s, xref, line, field,
                           H3/factor[UNITSh3], HT2/factor[UNITSh2],
                           lengthset, enforce, control);
    end
end;

procedure itemTHETA1B (dowhat: message);
const field= 8;
    min= 0;
    max= 100;
    openingset: scalarset= [percent];
    enforce= true;

begin
    with worknode^ .system do
    case dowhat of

        put_name: align ('THETA1=', xref, line);
        put_data: putquantity (THETA1B, UNITStheta1, xref, line, field);
        get_data: thequantity (THETA1B, UNITStheta1, xref, line, field,
            min, max, openingset, enforce, control);

    end
    end;

procedure itemTHETA2B (dowhat: message);

const field= 8;
    min= 0;
    max= 100;
    openingset: scalarset= [percent];
    enforce= true;

begin
    with worknode^ .system do
    case dowhat of

        put_name: align ('THETA2=', xref, line);
        put_data: putquantity (THETA2B, UNITStheta2, xref, line, field);
        get_data: thequantity (THETA2B, UNITStheta2, xref, line, field,
            min, max, openingset, enforce, control);

    end
    end;

procedure itemFLB (dowhat: message);

const field= 8;
    min= 0;
    max= 999.990;
    flowset: scalarset= [cfs,cfm,cfh,lps,lpm,lph];
    enforce= true;

begin
    with worknode^ .system do
```pascal
procedure putbox;

const  y= true;
   n= false;

var item: integer;

begin

header (put_data,top,left,right,'Design conditions');
textbox (left,top,right,bottom,n,y,y,y,0,0,1,1);
for item:= 0 to 3 do begin
    line:= topline+item;
    case item of
        0: itemH2SB (put_name);
        1: itemTHETA1B (put_name);
        2: itemTHETA2B (put_name);
        3: itemFLB (put_name);
    end
end
end;

procedure rubbox;

begin

window (left,top,right,bottom);
clrscr;
window (1,1,80,25);
end;

procedure putdata;

var item: integer;

begin
```
for item:= 0 to 3
  do begin
    line:= topline+item;
    case item of
      0: itemH2SB (put_data);
      1: itemTHETA1B (put_data);
      2: itemTHETA2B (put_data);
      3: itemFLB (put_data);
    end
  end
end;

procedure rubdata;
begin
  window (xref,topline,right-1,bottom-1);
  clrscr;
  window (1,1,60,25);
end;

procedure getdata;
begin
  case control of
  cr : item:= 0;
  home: item:= 3;
  endk: item:= 0;
  ccr : item:= 0;
end;
repeat
  line:= topline+item;
  case item of
      0: itemH2SB (get_data);
      1: itemTHETA1B (get_data);
      2: itemTHETA2B (get_data);
      3: itemFLB (get_data);
    end;
  case control of
  cr : item:= item+1;
  home: item:= item-1;
  endk: item:= item+1;
end;
end;
until not (item in [0..3])
or (control in menuset)
or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin
  case dowhat of
    put_box : putbox;
    rub_box : rubbox;
    put_data: putdata;
    rub_data: rubdata;
    get_data: getdata;
    end
  end; {designconditions}

begin {pagel}
  control:= null;
  nodeheader (put_item);
  pagenumber (put_box);
  pagenumber (put_item);
  menufooter (put_item);

  controls (put_box);
  physicalspecs (put_box);
  designconditions (put_box);
  repeat
    nodeheader (put_item);
    controls (put_item);
    physicalspecs (put_data);
    designconditions (put_data);
  repeat
    case box of
      1: nodeheader (get_data);
      2: controls (get_data);
      3: physicalspecs (get_data);
      4: designconditions (get_data);
    end;
    case control of
      cr : box:= box mod 4 + 1;
      home: if box= 1 then box:= 4 else box:= box-1;
      endk: box:= box mod 4 + 1;
ccr : box := box \mod 4 + 1;
end;
until (control in [uarr,darr,pgup,pgdn,esc])
or (control in menuset);
if (control in [uarr,darr])
then begin
  nodeheader (rub_item);
  controls (rub_item);
  physicalspecs (rub_data);
  designconditions (rub_data);
  case control of
    uarr: nodeup;
    darr: nodedown;
  end;
end;
until (control in [pgup,pgdn,esc])
or (control in menuset);
controls (rub_box);
physicalspecs (rub_box);
designconditions (rub_box);
end; (page1)
procedure page2 (var box, item: integer);

var line: integer;
    doval: boolean;
    dot0 : boolean;
    dot1 : boolean;
    doSP : boolean;
    doSP1: boolean;
    doSP2: boolean;
    doW : boolean;

    DISP: array[0..7] of integer;

procedure SETdesn;

begin
    doval:= true;
    dot0 := false;
    dot1 := false;
    doSP := false;
    doSP1:= false;
    doSP2:= false;
    doW := false;
end;

procedure SETstep;

begin
    doval:= false;
    dot0 := true;
    dot1 := false;
    doSP := true;
    doSP1:= false;
    doSP2:= false;
    doW := false;
end;

procedure SETpuls;

begin
    doval:= false;
dot0 := true;
dot1 := true;
doSP := true;
doSP1 := false;
doSP2 := false;
doW := false;
end;

procedure SETramp;
begin
doval := false;
dot0 := true;
dot1 := true;
doSP := false;
doSP1 := true;
doSP2 := true;
doW := false;
end;

procedure SETsine;
begin
doval := false;
dot0 := true;
dot1 := true;
doSP := true;
doSP1 := false;
doSP2 := false;
doW := true;
end;

procedure DISPdesn;
begin
DISP[0] := 0;
DISP[1] := 1;
end;

procedure DISPstep;
begin
DISP[0] := 0;
DISP[2] := 1;
DISP[4] := 2;
end;
procedure DISPpuls;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
end;

procedure DISPramp;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
  DISP[4]:= 4;
end;

procedure DISP sine;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
  DISP[4]:= 4;
end;

procedure H25profile (dowhat: message);
begin
  with worknodeA.system

const left= 10;
  xref= 22;
  right= 37;

  top= 5;
  topline= 7;
  lines= 7;
  bottom= 12;

procedure newitem;
begin
  with worknode^system
do begin
  case PROFILEh2s of
    DESN: SETdesn;
    STEP: SETstep;
    PULS: SETpuls;
    RAMP: SETramp;
    SINE: SETsine;
  end;
  case PROFILEh2s of
    DESN: DISPdesn;
    STEP: DISPstep;
    PULS: DISPpuls;
    RAMP: DISPramp;
    SINE: DISP sine;
  end;
end;

procedure itemPROFILEh2s (dowhat: message);
const field= 4;
  profileset: scalarset= [DESN,STEP,PULS,RAMP,SINE];
begin
  with worknode\$system do
  case dowhat of
    put_name: align ('profile= ',xref,line);
    put_data: putsymbol (PROFILEh2s,xref,line,field);
    get_data: thesymbol (PROFILEh2s,xref,line,field,
                           profileset,control);
  end
end;

procedure itemH2SB (dowhat: message);
const field= 8;
  (min= H3)
  (max= HT2)
  lengthset: scalarset= [inches,ft,cm,m];
enforce= true;
begin
with worknode^\.system do
  case dowhat of
    put_name: align ('H2s= ',xref,line);
    put_data: putquantity (H2SB,UNITSh2s,xref,line,field);
    get_data: thequantity (H2SB,UNITSh2s,xref,line,field,
      H3/factor[UNITSh3],HT2/factor[UNITSh2],
      lengthset,enforce,control);
  end
end;

procedure itemt0h2s (dowhat: message);

const field= 8;
  min= 0;
  max= 1440;
  timeset: scalarset= [sec,minutes,hrs];
  enforce= true;
begin
  with worknode^\.system do
    case dowhat of
      put_name: align ('t0= ',xref,line);
      put_data: putquantity (t0h2s,UNITSt0h2s,xref,line,field);
      get_data: thequantity (t0h2s,UNITSt0h2s,xref,line,field,
        min,max,timeset,enforce,control);
    end
  end;
end;

procedure itemt1h2s (dowhat: message);

const field= 8;
  min= 0;
  max= 1440;
  timeset: scalarset= [sec,minutes,hrs];
  enforce= true;
begin
  with worknode^\.system do
    case dowhat of
      put_name: align ('t1= ',xref,line);
      put_data: putquantity (t1h2s,UNITSt1h2s,xref,line,field);
      get_data: thequantity (t1h2s,UNITSt1h2s,xref,line,field,
procedure itemSPh2s (dowhat: message);

const field= 8;
  (min= H3)
  (max= HT2)
lengthset: scalarset= [inches,ft,cm,m];
enforce= true;

begin
  with Worknode\^\system do
    case dowhat of
      put_name: align ('SP= ',xref,line);
      put_data: putquantity (SPh2s,UNITSsph2s,xref,line,field);
      get_data: thequantity (SPh2s,UNITSsph2s,xref,line,field,
        H3/factor[UNITSh3],HT2/factor[UNITSht2],
        lengthset,enforce,control);
    end
  end;
end;
end;

procedure itemSP1h2s (dowhat: message);

const field= 8;
  (min= H3)
  (max= HT2)
lengthset: scalarset= [inches,ft,cm,m];
enforce= true;

begin
  with Worknode\^\system do
    case dowhat of
      put_name: align ('SP1= ',xref,line);
      put_data: putquantity (SP1h2s,UNITSsp1h2s,xref,line,field);
      get_data: thequantity (SP1h2s,UNITSsp1h2s,xref,line,field,
        H3/factor[UNITSh3],HT2/factor[UNITSht2],
        lengthset,enforce,control);
    end
  end;
end;
end;
end;

procedure itemSP2h2s (dowhat: message);
const field = 8;
(min = H3)
(max = HT2)
lengthset: scalarset = [inches, ft, cm, m];
enforce = true;

begin
  with worknode^ . system do
  case do what of
    put_name: align ('SP2=', xref, line);
    put_data: putquantity (SP2h2s, UNITSsp2h2s, xref, line, field);
    get_data: thequantity (SP2h2s, UNITSsp2h2s, xref, line, field,
      H3 / factor [UNITSh3], HT2 / factor [UNITSh2],
      lengthset, enforce, control);
  end
end;

procedure itemWh2s (do what: message);
const field = 8;
min = 0;
max = 999.990;
rateset: scalarset = [cps, cpm, cph];
enforce = true;

begin
  with worknode^ . system do
  case do what of
    put_name: align ('W=', xref, line);
    put_data: putquantity (Wh2s, UNITSwh2s, xref, line, field);
    get_data: thequantity (Wh2s, UNITSwh2s, xref, line, field,
      min, max, rateset, enforce, control);
  end
end;

procedure putbox;
const y = true;
n = false;
var item: integer;
begin
header (put_data, top, left, right, 'H2s profile');
textbox (left, top, right, bottom, n, y, y, 0, 0, 1, 1);
line := topline;
itemPROFILEh2s (put_name);
end;

procedure rubbox;
begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;
var item: integer;
begin
  for item := 0 to 7
    do begin
      line := topline + DISP(item);
      case item of
        0: itemPROFILEh2s (put_data);
        1: if doval
           then begin
               itemH2SB (put_name);
               itemH2SB (put_data);
           end;
        2: if dot0
           then begin
               item0h2s (put_name);
               item0h2s (put_data);
           end;
        3: if dot1
           then begin
               item1h2s (put_name);
               item1h2s (put_data);
           end;
        4: if doSP
           then begin
               itemSPh2s (put_name);
               itemSPh2s (put_data);
           end;
        5: if doSP1
           then begin
               itemSP1h2s (put_name);
           end;
    end;
end;
itemSP1h2s (put_data);
end;

6: if doSP2
    then begin
        itemSP2h2s (put_name);
        itemSP2h2s (put_data);
    end;

7: if doW
    then begin
        itemW2h2s (put_name);
        itemW2h2s (put_data);
    end;
end;

procedure rubitem;
begin
    window <xref, topline, right-1, topline+1>;
gotoxy (1, 1);
clear;
    window (left+1, topline+1, right-1, bottom-1);
clearscr;
    window (1, 1, 80, 25);
end;

procedure getdata;

procedure getPROFILEh2s;

var oldPROFILEh2s: scalar;

begin
    with worknode^system
    do begin
        oldPROFILEh2s := PROFILEh2s;
        itemPROFILEh2s (get_data);
        if (PROFILEh2s<>oldPROFILEh2s)
            then begin
                rubitem;
                newitem;
                putitem;
            end;
    end;
end;
begin
  case control of
    cr  : item:= 0;
    home: item:= 6;
    endk: item:= 0;
    ccr : item:= 0;
    uarr: item:= 0;
    darr: item:= 0;
  end;
repeat
  line:= topline+DISP[item];
  case item of
    0: getPROFILEh2s;
    1: if doval then itemH2SB = get_data;
    2: if dot0 then item0h2s = get_data;
    3: if dotl then item1h2s = get_data;
    4: if doSP then itemSPh2s = get_data;
    5: if doSPI then itemSPIh2s = get_data;
    6: if doSP2 then itemSP2h2s = get_data;
    7: if doW  then itemWh2s = get_data;
  end;
  case control of
    cr  : item:= item+1;
    home: item:= item-1;
    endk: item:= item+1;
  end;
  until not (item in [0..7])
    or (control in menuSet)
    or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (PROFILEh2s)
  newitem;
  case dowhat of
    put_box : putbox;
    rub_box : rubbox;
    put_item: putitem;
    rub_item: rubitem;
    get_data: getdata;
  end
procedure THETA1profile (dowhat: message);

const left = 43;
  xref = 55;
  right = 70;
  top = 5;
  topline = 7;
  lines = 5;
  bottom = 12;

procedure newitem;
begin
  with worknode^.system do begin
    case PROFILEtheta1 of
      DESN: SETdesn;
      STEP: SETstep;
      PULS: SETpuls;
      RAMP: SETramp;
      SINE: SETsine;
    end;
  end;
  case PROFILEtheta1 of
    DESN: DISPdesn;
    STEP: DISPstep;
    PULS: DISPpuls;
    RAMP: DISPramp;
    SINE: DISPsine;
  end;
end;
end;

procedure itemPROFILEtheta1 (dowhat: message);
const field = 4;
  profileset: scalarset = [DESN, STEP, PULS, RAMP, SINE];
begin
  with worknode^.system do begin
    case dowhat of

put_name: align ('profile=', xref, line);
put_data: putsymbol (PROFILEthetal, xref, line, field);
get_data: thesymbol (PROFILEthetal, xref, line, field, profileset, control);
end
end;

procedure itemTHETA1B (dowhat: message);
const field= 8;
  min= 0;
  max= 100;
  openingset: scalarset= [percent];
  enforce= true;
begin
  with worknode^.system do
      case dowhat of
      put_name: align ('THETAl=', xref, line);
      put_data: putquantity (THETAlB, UNITStotal, xref, line, field);
      get_data: thequantity (THETAlB, UNITStotal, xref, line, field, min, max, openingset, enforce, control);
      end
      end;

procedure itemtOthetal (dowhat: message);
const field= 8;
  min= 0;
  max= 1440;
  timeset: scalarset= [sec, minutes, hrs];
  enforce= true;
begin
  with worknode^.system do
      case dowhat of
      put_name: align ('tO=', xref, line);
      put_data: putquantity (tOtetal, UNITStotal, xref, line, field);
      get_data: thequantity (tOtetal, UNITStotal, xref, line, field, min, max, timeset, enforce, control);
      end
      end;
procedure itemt1theta1 (dowhat: message);

const field= 8;
  min= 0;
  max= 1440;
  timeset: scalarset= [sec, minutes, hrs];
  enforce= true;
begin
  with worknode^.system do
    case dowhat of
      put_name: align ("t1= ", xref, line);
      put_data: putquantity (t1theta1, UNITSt1theta1, xref, line, field);
      get_data: thequantity (t1theta1, UNITSt1theta1, xref, line, field,
        min, max, timeset, enforce, control);
    end
  end;
end

procedure itemSPtheta1 (dowhat: message);

const field= 8;
  min= 0;
  max= 100;
  openingset: scalarset= [percent];
  enforce= true;
begin
  with worknode^.system do
    case dowhat of
      put_name: align ("SP= ", xref, line);
      put_data: putquantity (SPtheta1, UNITSptheta1, xref, line, field);
      get_data: thequantity (SPtheta1, UNITSptheta1, xref, line, field,
        min, max, openingset, enforce, control);
    end
  end;
end

procedure itemSPltheta1 (dowhat: message);

const field= 8;
  min= 0;
max = 100;
openingset: scalarset = [percent];
enforce = true;

begin
  with worknode^,system do
  case dowhat of
      put_name: align ('SP1=', xref, line);
      put_data: putquantity (SP1theta1, UNITSsp1theta1, xref, line, field);
      get_data: thequantity (SP1theta1, UNITSsp1theta1, xref, line, field,
        min, max, openingset, enforce, control);
  end
end;

procedure itemSP2theta1 (dowhat: message);
const field = 8;
  min = 0;
  max = 100;
openingset: scalarset = [percent];
enforce = true;

begin
  with worknode^,system do
  case dowhat of
      put_name: align ('SP2=', xref, line);
      put_data: putquantity (SP2theta1, UNITSsp2theta1, xref, line, field);
      get_data: thequantity (SP2theta1, UNITSsp2theta1, xref, line, field,
        min, max, openingset, enforce, control);
  end
end;

procedure itemWtheta1 (dowhat: message);
const field = 8;
  min = 0;
  max = 999.990;
rateset: scalarset = [cps, cpm, cph];
enforce = true;

begin
  with worknode^,system do
  case dowhat of
put_name: align ('W= ',xref,line);
put_data: putquantity (Wthetal,UNITSwtheta1,xref,line,field);
get_data: thequantity (Wthetal,UNITSwtheta1,xref,line,field,
  min,max,rateset,enforce,control);

end
end;

procedure putbox;
const y= true;
  n= false;
var item: integer;

begin
  header (put_data,top,left,right,'THETA1 profile');
  textbox (left,top,right,bottom,n,y,y,y,0,0,1,1);
  line:= topline;
  itemPROFILEthetal (put_name);
end;

procedure rubbox;

begin
  window (left,top,right,bottom);
  clrscr;
  window (1,1,80,25);
end;

procedure putitem;

var item: integer;

begin
  for item:= 0 to 7
    do begin
      line:= topline+DISP(item);
      case item of
        0: itemPROFILEthetal (put_data);
        1: if doval
          then begin
            itemTHETA1B (put_name);
            itemTHETA1B (put_data);
          end;
2: if dot0
then begin
  item0thetal (put_name);
  item0thetal (put_data);
end;
3: if dot1
then begin
  item1thetal (put_name);
  item1thetal (put_data);
end;
4: if doSP
then begin
  itemSPthetal (put_name);
  itemSPthetal (put_data);
end;
5: if doSP1
then begin
  itemSP1thetal (put_name);
  itemSP1thetal (put_data);
end;
6: if doSP2
then begin
  itemSP2thetal (put_name);
  itemSP2thetal (put_data);
end;
7: if doW
then begin
  itemWthetal (put_name);
  itemWthetal (put_data);
end;
end;
end;
end;

procedure rubitem;
begin
  window (xref, topline, right-1, topline+1);
gotoxy (1,1);
creol;
  window (left+1, topline+1, right-1, bottom-1);
crscr;
  window (1,1, 80, 25);
end;

procedure getdata;
procedure getPROFILEthetal;

var oldPROFILEthetal: scalar;

begin
  with worknode^.system
  do begin
    oldPROFILEthetal := PROFILEthetal;
    itemPROFILEthetal (get_data);
    if (PROFILEthetal()oldPROFILEtheta1J
      then begin
        rubitem;
        newitem;
        putitem;
      end;
    end;
  end;

begin
  case control of
  cr : item:= 0;
  home: item:= &;
  endk: item:= 0;
  ccr : item:= 0;
  uarr: item:= 0;
  darr: item:= 0;
  end;
repeat
  lines:= topline+DISP[item];
  case item of
  0: getPROFILEthetal;
  1: if doval then itemTHETA1B (get_data);
  2: if dot0 then item0thetal (get_data);
  3: if dot1 then item1thetal (get_data);
  4: if doSP then itemSPthetal (get_data);
  5: if doSP1 then itemSP1thetal (get_data);
  6: if doSP2 then itemSP2thetal (get_data);
  7: if doW then itemWthetal (get_data);
end;
  case control of
  cr : item:= item+1;
  home: item:= item-1;
  endk: item:= item+1;
end;
   until not (item in [0..7])
      or (control in menuset)
      or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (PROFILEtheta1)
   newitem;
   case dowhat of
      put_box : putbox;
      rub_box : rubbox;
      put_item : putitem;
      rub_item: rubitem;
      get_data: getdata;
   end
end;

procedure THETA2profile (dowhat: message);
const left= 10;
   xref= 22;
   right= 37;
   top= 14;
   topline= 16;
   lines= 5;
   bottom= 21;

procedure newitem;
   begin
      with worknode^.system do begin
         case PROFILEtheta2 of
            DESN: SETdesn;
            STEP: SETstep;
            PULS: SETpuls;
            RAMP: SETramp;
            SINE: SETsine;
         end;
      end;
   case PROFILEtheta2 of
      DESN: DISPdesn;
   end;
STEP: DISPstep;
PULS: DISPpuls;
RAMP: DISPra1p;
SINE: DISPsine;

end;
end;
end;

procedure itemPROFILEtheta2 (dowhat: message);
const field= 4;
profileset: scalarset= [DESN,STEP,PULS,RAMP,SINE];
begin
  with worknode^.system do
  case dowhat of
  put_name: align ('profile= ',xref,line);
  put_data: putsymbol (PROFILEtheta2,xref,line,field);
  get_data: thesymbol (PROFILEtheta2,xref,line,field,
    profileset,control);
  end
end;

procedure itemTHETA2B (dowhat: message);
const field= 8;
min= 0;
max= 100;
openingset: scalarset= [percent];
enforce= true;
begin
  with worknode^.system do
  case dowhat of
  put_name: align ('THETA2= ',xref,line);
  put_data: putquantity (THETA2B,UNITStheta2,xref,line,field);
  get_data: thequantity (THETA2B,UNITStheta2,xref,line,field,
    min,max,openingset,enforce,control);
  end
end;
procedure itemt0theta2 (dowhat: message);

const field= 8;
min= 0;
max= 1440;
timeset: scalarset= [sec,minutes,hrs];
enforce= true;

begin
with worknode^.system do
  case dowhat of
    put_name: align ('t0= ',xref,line);
    put_data: putquantity (t0theta2,UNITSt0theta2,xref,line,field);
    get_data: thequantity (t0theta2,UNITSt0theta2,xref,line,field,
                           min,max,timeset,enforce,control);
  end
end;

procedure itemt1theta2 (dowhat: message);

const field= 8;
min= 0;
max= 1440;
timeset: scalarset= [sec,minutes,hrs];
enforce= true;

begin
with worknode^.system do
  case dowhat of
    put_name: align ('t1= ',xref,line);
    put_data: putquantity (t1theta2,UNITSt1theta2,xref,line,field);
    get_data: thequantity (t1theta2,UNITSt1theta2,xref,line,field,
                           min,max,timeset,enforce,control);
  end
end;

procedure itemSPtheta2 (dowhat: message);

const field= 8;
min= 0;
max= 100;
openingset: scalarset= [percent];
enforce= true;
begin
  with worknode\textasciitilde.system do
  case dowhat of
    put_name: align (SP\textasciitilde,xref,line);
    put_data: putquantity (SP\textasciitilde,UNITS\textasciitilde,xref,line,field);
    get_data: thequantity (SP\textasciitilde,UNITS\textasciitilde,xref,line,field,
      min,max,openingset,enforce,control);
  end
end;

procedure itemSP1\textasciitilde (dowhat: message);
const field= 8;
  min= 0;
  max= 100;
  openingset: scalarset= \{percent\};
enforce= true;
begin
  with worknode\textasciitilde.system do
  case dowhat of
    put_name: align (SP1\textasciitilde,xref,line);
    put_data: putquantity (SP1\textasciitilde,UNITS\textasciitilde,xref,line,field);
    get_data: thequantity (SP1\textasciitilde,UNITS\textasciitilde,xref,line,field,
      min,max,openingset,enforce,control);
  end
end;

procedure itemSP2\textasciitilde (dowhat: message);
const field= 8;
  min= 0;
  max= 100;
  openingset: scalarset= \{percent\};
enforce= true;
begin
  with worknode\textasciitilde.system do
  case dowhat of
    put_name: align (SP2\textasciitilde,xref,line);
    put_data: putquantity (SP2\textasciitilde,UNITS\textasciitilde,xref,line,field);
    get_data: thequantity (SP2\textasciitilde,UNITS\textasciitilde,xref,line,field,
procedure itemWtheta2 (dowhat: message);
const
field = 8;
min = 0;
max = 999.990;
rateset: scalarset = [cps, cpm, cph];
enforce = true;

begin
  with worknode^ . system do
    case dowhat of
      put_name: align ('W= ', xref, line);
      put_data: putquantity (Wtheta2, UNITSWtheta2, xref, line, field);
      get_data: thequantity (Wtheta2, UNITSWtheta2, xref, line, field,
                              min, max, rateset, enforce, control);
    end
  end;
end;

procedure putbox;

const
y = true;
n = false;

var
item: integer;

begin
  header (put_data, top, left, right, 'THETA2 profile');
  textbox (left, top, right, bottom, n, y, y, y, 0, 0, 1, 1);
  line := topline;
  itemPROFILEtheta2 (put_name);
end;

procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;
procedure putitem;

var item: integer;

begin
  for item:= 0 to 7
  do begin
    line:= topline+DISP[item];
    case item of
      0: itemPROFILEtheta2 (put_data);
      1: if doval
          then begin
              itemTHETA2B (put_name);
              itemTHETA2B (put_data);
            end;
      2: if dot0
          then begin
              item0theta2 (put_name);
              item0theta2 (put_data);
            end;
      3: if dot1
          then begin
              item1theta2 (put_name);
              item1theta2 (put_data);
            end;
      4: if doSP
          then begin
              itemSPtheta2 (put_name);
              itemSPtheta2 (put_data);
            end;
      5: if doSP1
          then begin
              itemSP1theta2 (put_name);
              itemSP1theta2 (put_data);
            end;
      6: if doSP2
          then begin
              itemSP2theta2 (put_name);
              itemSP2theta2 (put_data);
            end;
      7: if doW
          then begin
              itemWtheta2 (put_name);
              itemWtheta2 (put_data);
            end;
    end;
  end;
procedure rubitem;
begin
  window (xref, topline, right-1, topline+1);
  gotoxy (1,1);
  clrreol;
  window (left+1, topline+1, right-1, bottom-1);
  clrscr;
  window (1,1,80,25);
end;

procedure getdata;

procedure getPROFILEtheta2;
var oldPROFILEtheta2: scalar;
begin
  with worknode^system do begin
    oldPROFILEtheta2:= PROFILEtheta2;
    itemPROFILEtheta2 (get_data);
    if (PROFILEtheta2<>oldPROFILEtheta2)
      then begin
        rubitem;
        newitem;
        putitem;
      end;
  end;
end;

begin
  case control of
    cr : item:= 0;
    home: item:= 6;
    endk: item:= 0;
    ccr : item:= 0;
    uarr: item:= 0;
    darr: item:= 0;
  end;
repeat
  line:= topline+DISP[item];
case item of
0: getPROFILEtheta2;
1: if doval then itemTHETA2B (get_data);
2: if dot0 then item0theta2 (get_data);
3: if dot1 then item1theta2 (get_data);
4: if doSP then itemSPtheta2 (get_data);
5: if doSP1 then itemSP1theta2 (get_data);
6: if doSP2 then itemSP2theta2 (get_data);
7: if doW then itemWtheta2 (get_data);
end;
case control of
   cr : item:= item+1;
   home: item:= item-1;
   endk: item:= item+1;
end;
until not (item in [0..7])
   or (control in menuset)
   or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;
begin {PROFILEtheta2}
newitem;
case dowhat of
   put_box : putbox;
   rub_box : rubbox;
   put_item: putitem;
   rub_item: rubitem;
   get_data: getdata;
end
end;

procedure FLprofile (dowhat: message);
const left= 43;
   xref= 55;
   right= 70;
   top= 14;
   topline= 16;
   lines= 5;
   bottom= 21;
procedure newitem;
begin
with worknode^.system do begin
  case PROFILEfl of
    DESN: SETdesn;
    STEP: SETstep;
    PULS: SETpuls;
    RAMP: SETramp;
    SINE: SETsine;
  end;
  case PROFILEfl of
    DESN: DISPdesn;
    STEP: DISPstep;
    PULS: DISPpuls;
    RAMP: DISPramp;
    SINE: DISPsine;
  end;
end;
end;
end;

procedure itemPROFILEfl (dowhat: message);
const field= 4;
  profileset: scalarset= [DESN,STEP,PULS,RAMP,SINE];
begin
  with worknode^.system do
    case dowhat of
      put_name: align ('profile= ','xref,line);
      put_data: putsymbol (PROFILEfl,xref,line,field);
      get_data: thesymbol (PROFILEfl,xref,line,field,
                              profileset,control);
    end
  end;

procedure itemFLB (dowhat: message);
const field= 8;
min = 0;
max = 999.990;
flowset: scalarset = [cfs,cfm,cfh,lps,lpm,lph];
enforce= true;

begin
with worknode^ .system do
    case dowhat of
        put_name: align ('FL= ',xref,line);
        put_data: putquantity (FLB,UNITSt0f1,xref,line,field);
        get_data: thequantity (FLB,UNITSt0f1,xref,line,field,
            min,max,flowset,enforce,control);
    end
end;

procedure itemt0fl (dowhat: message);
const field= 8;
    min = 0;
    max = 1440;
    timeset: scalarset = [sec,minutes,hrs];
enforce= true;
begin
    with worknode^ .system do
        case dowhat of
            put_name: align ('T0= ',xref,line);
            put_data: putquantity (T0f1,UNITSt0f1,xref,line,field);
            get_data: thequantity (T0f1,UNITSt0f1,xref,line,field,
                min,max,timeset,enforce,control);
        end
    end;

procedure itemt1fl (dowhat: message);
const field= 8;
    min = 0;
    max = 1440;
    timeset: scalarset = [sec,minutes,hrs];
enforce= true;
begin
    with worknode^ .system do
case dowhat of
  put_name: align ("tl= ',xref,line);
  put_data: putquantity (tlfl,UNITStlfl,xref,line,field);
  get_data: thequantity (tlfl,UNITStlfl,xref,line,field,
    min,max,flowset,enforce,control);
end
end;

procedure itemSPfl (dowhat: message);

const field= 8;
  min= 0;
  max= 999.990;
  flowset: scalarset= [inches,ft,cm,m];
  enforce= true;
begin
  with worknodeA.system do
    case dowhat of
      put_name: align ('SP= ',xref,line);
      put_data: putquantity (SPfl,UNITSpfl,xref,line,field);
      get_data: thequantity (SPfl,UNITSpfl,xref,line,field,
        min,max,flowset,enforce,control);
    end
end;

procedure itemSP1fl (dowhat: message);

const field= 8;
  min= 0;
  max= 999.990;
  flowset: scalarset= [cfs,cfh,cf,lps,lpm,lph];
  enforce= true;
begin
  with worknodeA.system do
    case dowhat of
      put_name: align ('SP1= ',xref,line);
      put_data: putquantity (SP1fl,UNITSp1fl,xref,line,field);
      get_data: thequantity (SP1fl,UNITSp1fl,xref,line,field,
        min,max,flowset,enforce,control);
    end
end;
procedure itemSP2fl (dowhat: message);

const field= 8;
  min= 0;
  max= 999.990;
  flowset: scalarset= [cfs,cfm,cfh,lps,lpm,lph];
  enforce= true;

begin
  with worknode^.system do
    case dowhat of
    put_name: align ('SP2= ',xref,line);
    put_data: putquantity (SP2fl,UNITSp2fl,xref,line,field);
    get_data: thequantity (SP2fl,UNITSp2fl,xref,line,field,
      min,max,flowset,enforce,control);
    end
  end;
end;

procedure itemWfl (dowhat: message);

const field= 8;
  min= 0;
  max= 999.990;
  rateset: scalarset= [cps,cpm,cph];
  enforce= true;

begin
  with worknode^.system do
    case dowhat of
    put_name: align ('W= ',xref,line);
    put_data: putquantity (Wfl,UNITSwfl,xref,line,field);
    get_data: thequantity (Wfl,UNITSwfl,xref,line,field,
      min,max,rateset,enforce,control);
    end
  end;
end;

procedure putbox;

const y= true;
  n= false;
var item: integer;

begin
  header (put_data,top,left,right,'FL profile');
  textbox (left,top,right,bottom,n,y,y,0,0,1,1);
  line:= topline;
  itemPROFILEf1 (put_name);
end;

procedure rubbox;
begin
  window (left,top,right,bottom);
  clrscr;
  window (1,1,80,25);
end;

procedure putitem;
var item: integer;

begin
  for item:= 0 to 7 do begin
    line:= topline+DISP[item];
    case item of
    0: itemPROFILEf1 (put_data);
    1: if doval then begin
        itemFLB (put_name);
        itemFLB (put_data);
      end;
    2: if dot0 then begin
        itemt0fl (put_name);
        itemt0fl (put_data);
      end;
    3: if dot1 then begin
        itemt1fl (put_name);
        itemt1fl (put_data);
      end;
    4: if doSP then begin
        itemSPfl (put_name);
        itemSPfl (put_data);
      end;
  end;
end;
5: if doSP1
  then begin
    itemSP1f1 (put_name);
    itemSP1f1 (put_data);
  end;
6: if doSP2
  then begin
    itemSP2f1 (put_name);
    itemSP2f1 (put_data);
  end;
7: if doW
  then begin
    itemWf1 (put_name);
    itemWf1 (put_data);
  end;
end;
end;

procedure rubitem;
begin
  window (xref, topline, right-1, topline+1);
  gotoxy (1,1);
  clrscr;
  window (left+1, topline+1, right-1, bottom-1);
  clrscr;
  window (1,1,80,25);
end;

procedure getdata;

procedure getPROFILEf1;
var oldPROFILEf1: scalar;
begin
  with worknode^.system do begin
    oldPROFILEf1 := PROFILEf1;
    itemPROFILEf1 (get_data);
    if (PROFILEf1 <> oldPROFILEf1) then begin
      rubitem;
      newitem;
      putitem;
    end;
end;
end;

begin
  case control of
    cr : item:= 0;
    home: item:= 6;
    endk: item:= 0;
    ccr : item:= 0;
    uarr: item:= 0;
    darr: item:= 0;
  end;
  repeat
    line:= topline+DISP(item);
    case item of
      0: getPROFILEfl;
      1: if doval then itemFLB (get_data);
      2: if dot0 then item0fl (get_data);
      3: if dot1 then item1fl (get_data);
      4: if doSP then itemSPfl (get_data);
      5: if doSP1 then itemSP!fl (get_data);
      6: if doSP2 then itemSP2f1 (get_data);
      7: if doW then itemWf1 (get_data);
    end;
    case control of
      cr : item:= item+1;
      home: item:= item-1;
      endk: item:= item+1;
    end;
    until not (item in [0..7])
      or (control in menuiset)
      or (control in [ccr,pup,pgdn,uarr,darr,esc]);
  end;

begin (PROFILEfl)
  newitem;
  case dowhat of
    put_box : putbox;
    rub_box : rubbox;
    put_item: putitem;
    rub_item: rubitem;
  end;
begin {page2}
  control:= null;
  nodeheader (put_item);
  pagenumber (put_box);
  pagenumber (put_item);
  menufooter (put_item);

H2Sprofile (put_box);
THETA1profile (put_box);
THETA2profile (put_box);
FLprofile (put_box);
repeat
  nodeheader (put_item);
  H2Sprofile (put_item);
  THETA1profile (put_item);
  THETA2profile (put_item);
  FLprofile (put_item);
repeat
  case box of
    1: nodeheader (get_data);
    2: H2Sprofile (get_data);
    3: THETA1profile (get_data);
    4: THETA2profile (get_data);
    5: FLprofile (get_data);
  end;
  case control of
    cr : box:= box mod 5 + 1;
    home: if box = 1 then box:= 5 else box:= box-1;
    endk: box:= box mod 5 + 1;
    ccr : box:= box mod 5 + 1;
  end;
  until (control in [uarr,darr,pgup,pgdn,esc])
  or  (control in menuset);
  if  (control in [uarr,darr])
    then begin
      nodeheader (rub_item);
      H2Sprofile (rub_item);
      THETA1profile (rub_item);
      THETA2profile (rub_item);
      FLprofile (rub_item);
    end;
end;
case control of

    uarr: nodeup;
    darr: nodedown;

end;
end;

until (control in [pgup,pgdn,esc])
or  (control in menuset);
H2Sprofile  (rub_box);
THETA1profile (rub_box);
THETA2profile (rub_box);
FLprofile   (rub_box);
end; {page2}
procedure page3(var box, item: integer);
var line: integer;

procedure PUMPPopcurve(dowhat: message);
const left = 5;
xref = 17;
right = 32;
top = 4;
topline = 6;
lines = 6;
bottom = 12;
var doBM: boolean;
doVMIN: boolean;
doVMAX: boolean;
doFMIN: boolean;
doFMAX: boolean;

DISP: array[0..5] of integer;

procedure SETlin;
begin
  doBM := false;
doVMIN := true;
doVMAX := true;
doFMIN := true;
doFMAX := true;
end;

procedure SETxpo;
begin
  doBM := true;
doVMIN := true;
doVMAX := true;
doFMIN := true;
doFMAX := true;
end;
procedure DISPlin;
begin
  DISP[0]:= 0;
  DISP[2]:= 1;
  DISP[3]:= 2;
  DISP[4]:= 3;
  DISP[5]:= 4;
end;

procedure DISPxpo;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
  DISP[3]:= 3;
  DISP[4]:= 4;
  DISP[5]:= 5;
end;

procedure newitem;
begin
  with worknode^ .system do begin
    case OPCURVEpump of
      LIN: SETlin;
      XPO: SETxpo;
    end;
    case OPCURVEpump of
      LIN: DISPlin;
      XPO: DISPxpo;
    end;
    end;
end;

procedure itemOPCURVEpump (dowhat: message);
const field= 3;
optcurveset: scalarset= [LIN,XPO];
begin
  with worknode-.system do
  case dowhat of
  
    put_name: align ('opcurve= ',xref,line);
    put_data: putsymbol (OPCURVEpump,xref,line,field);
    get_data: thesymbol (OPCURVEpump,xref,line,field, 
                        opcurveset,control);
  end
end;

procedure itemBM (dowhat: message);

const field= 8;
  min= -10;
  max= 10;
  nullset: scalarset= [none];
  enforce= true;

begin
  with worknode-.system do
  case dowhat of
  
    put_name: align ('Bm= ',xref,line);
    put_data: putquantity (BM,UNITSBM,xref,line,field);
    get_data: thequantity (BM,UNITSBM,xref,line,field, 
                           min,max,nullset,enforce,control);
  end
end;

procedure itemVMIN (dowhat: message);

const field= B;
  min= 0;
  {max= VMAX}
  voltageset: scalarset= [mv,v];
  enforce= true;

begin
  with worknode-.system do
  case dowhat of
  
    put_name: align ('Vmin= ',xref,line);
    put_data: putquantity (VHIN,UNITSVmin,xref,line,field);
    get_data: thequantity (VMIN,UNITSVmin,xref,line,field, 
                           min,VMAX/factor[UNITSV], 
                           voltageset,control,control);
  end
procedure itemVMAX (dowhat: message);

const field= 8;
  (min= VMIN)
  max= 999.99;
  voltageset: scalarset= [mv,v];
  enforce= true;

begin
  with worknode.A.system do
  case dowhat of
    put_name: align ('Vmax= ',xref,line);
    put_data: putquantity (VMAX,UNITSvmax,xref,line,field);
    get_data: thequantity (VMAX,UNITSvmax,xref,line,field,
                          VMIN/factor[UNITSvmin],max,
                          voltageset,enforce,control);
  end
end;

procedure itemFMIN (dowhat: message);

const field= 8;
  min= 0;
  (max= FIMAX)
  flowset: scalarset= [cfs,cfm,cfh,lps,lpm,lph];
  enforce= true;

begin
  with worknode.A.system do
  case dowhat of
    put_name: align ('F1min= ',xref,line);
    put_data: putquantity (F1MIN,UNITSfmin,xref,line,field);
    get_data: thequantity (F1MIN,UNITSfmin,xref,line,field,
                          min,FIMAX/factor[UNITSfmax],
                          flowset,enforce,control);
  end
end;

procedure itemFMAX (dowhat: message);

const field= 8;
  (min= FIMIN)
  max= 5000;
flowset: scalarset= [cfs,cfm,cfh,lps,lpm,lph];
enforce= true;

begin
  with worknode^.system do
  case dowhat of
    put_name: align ('F1max= ', xref, line);
    put_data: putquantity (F1MAX, UNITSf1max, xref, line, field);
    get_data: thequantity (F1MIN/factor[UNITSf1min], max,
                           flowset, enforce, control);
  end;
end;

procedure putbox;

const y = true;
  n = false;
var item: integer;

begin
  F1min/factor[UNITSf1ax, flowset, enforce, control];
  header (put_data, top, left, right, 'PUMP opcurve');
  textbox (left, top, right, bottom, n, y, y, 0, 0, 1, 1);
  line:= topline;
  itemOPCURVEpump (put_name);
end;

procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;

var item: integer;

begin
  for item:= 0 to 5
    do begin
      line:= topline+DISP[item];
      case item of
        end;
end;

begin
  with worknode^.system do
  case dowhat of
    put_name: align ('F1max= ', xref, line);
    put_data: putquantity (F1MAX, UNITSf1max, xref, line, field);
    get_data: thequantity (F1MIN/factor[UNITSf1min], max,
                           flowset, enforce, control);
  end;
end;

procedure putbox;

const y = true;
  n = false;
var item: integer;

begin
  F1min/factor[UNITSf1ax, flowset, enforce, control];
  header (put_data, top, left, right, 'PUMP opcurve');
  textbox (left, top, right, bottom, n, y, y, 0, 0, 1, 1);
  line:= topline;
  itemOPCURVEpump (put_name);
end;

procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;

var item: integer;

begin
  for item:= 0 to 5
    do begin
      line:= topline+DISP[item];
      case item of
        end;
end;
0: itemOPCURVEpump (put_data);
1: if doBM
   then begin
      itemBM (put_name);
      itemBM (put_data);
   end;
2: if doVMIN
   then begin
      itemVMIN (put_name);
      itemVMIN (put_data);
   end;
3: if doVMAX
   then begin
      itemVMAX (put_name);
      itemVMAX (put_data);
   end;
4: if doF1MIN
   then begin
      itemF1MIN (put_name);
      itemF1MIN (put_data);
   end;
5: if doF1MAX
   then begin
      itemF1MAX (put_name);
      itemF1MAX (put_data);
   end;
end;

procedure rubitem;
begin
   window (xref, topline, right-1, topline+1);
   gotoxy (1,1);
   clrrel;
   window (left+1, topline+1, right-1, bottom-1);
   clrsrm;
   window (1,1,80,25);
end;

procedure getdata;

procedure getOPCURVEpump;
var oldOPCURVEpump: scalar;
begin
with worknode'system
do begin
oldOPCURVEpump:= OPCURVEpump;
itemOPCURVEpump (get_data);
if (OPCURVEpump<>oldOPCURVEpump)
then begin
rubite;
newite;
pute;
end;
end;
end;

begin
case control of
  cr : item:= 0;
  home: item:= 5;
  endk: item:= 0;
  ccr : item:= 0;
  uarr: item:= 0;
  darr: item:= 0;
end;
repeat
  line:= topline+DISP[item];
  case item of
    0: getOPCURVEpump;
    1: if doBM then itemBM (get_data);
    2: if doVMIN then itemVMIN (get_data);
    3: if doVMAX then itemVMAX (get_data);
    4: if doFIMIN then itemFIMIN (get_data);
    5: if doFIMAX then itemFIMAX (get_data);
  end;
  case control of
    cr : item:= item+1;
    home: item:= item-1;
    endk: item:= item+1;
  end;
until not (item in [0..5])
  or (control in menuet)
  or (control in [ccr, pgup, pgdn, uarr, darr, esc]);
end;
begin {OFCURVEpump}
newitem;
case dowhat of

put_box : putbox;
rub_box : rubbox;
put_item : putitem;
rub_item : rubitem;
get_data : getdata;
end
end;

procedure VALVE1opcurve (dowhat: message);

const left= 37;
xref= 50;
right= 75;

top= 7;
topline= 9;
lines= 3;
bottom= 12;

var doB1 : boolean;
doCV1: boolean;

DISP: array[0..2] of integer;

procedure SETlin;

begin
  doB1 := false;
doCV1:= true;
end;

procedure SETxpo;

begin
  doB1 := true;
doCV1:= true;
end;

procedure DISPlin;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
end;

procedure DISPxpo;
begin
  DISP[0]:= 0;
  DISP[1]:= 1;
  DISP[2]:= 2;
end;

procedure newitem;
begin
  with worknode^.system do begin
    case OPCURVEvalvel of
      LIN: SETlin;
      XPO: SETxpo;
    end;
    case OPCURVEvalvel of
      LIN: DISPlin;
      XPO: DISPxpo;
    end;
  end;
end;

procedure itemOPCURVEvalvel (dowhat: message);
const
  field= 3;
  opcurveset: scalarset= [LIN,XPO];
begin
  with worknode^.system do
  case dowhat of
    put_name: align ('opcurve= ',xref,line);
    put_data: putsymbol (OPCURVEvalvel,xref,line,field);
    get_data: thesymbol (OPCURVEvalvel,xref,line,field,
                         opcurveset,control);
procedure itemB1 (dowhat: message);
begin
  with worknode^.system do
  case dowhat of
    put_name: align ('B1=', xref, line);
    put_data: putquantity (B1, UNITSb1, xref, line, field);
    get_data: thequantity (B1, UNITSb1, xref, line, field, min, max, nullset, enforce, control);
  end
end;

procedure itemCV1 (dowhat: message);
begin
  with worknode^.system do
  case dowhat of
    put_name: align ('CV1=', xref, line);
    put_data: putquantity (CV1, UNITScv1, xref, line, field);
    get_data: thequantity (CV1, UNITScv1, xref, line, field, min, max, nullset, enforce, control);
  end
end;

procedure putbox;
const y = true;
n= false;
var item: integer;
begin
  header (put_data,top,left,right,'VALVE 1 opcurve');
  textbox (left,top,right,bottom,n,y,y,0,0,1,1);
  line:= topline;
  itemOPCURVEvalvel (put_name);
end;

procedure rubbox;
begin
  window (left,top,right,bottom);
  clrscr;
  window (1,1,80,25);
end;

procedure putitem;
var item: integer;
begin
  for item:= 0 to 2
  do begin
    line:= topline+DISP(item);
    case item of
      0: itemOPCURVEvalvel (put_data);
      1: if doB1
        then begin
          itemB1 (put_name);
          itemB1 (put_data);
        end;
      2: if doCV1
        then begin
          itemCV1 (put_name);
          itemCV1 (put_data);
        end;
  end;
end;

procedure rubitem;
begin
procedure getdata;

procedure getOPCURVEvalvel;

var oldOPCURVEvalvel: scalar;

begin
  with worknode^.system do begin
    oldOPCURVEvalvel := OPCURVEvalvel;
    itemOPCURVEvalvel (get_data);
    if (OPCURVEvalvel <> oldOPCURVEvalvel) then begin
      rubitem;
      newitem;
      putitem;
    end;
  end;
end;

begin
  case control of
    cr : item := 0;
    home : item := 2;
    endk : item := 0;
    ccr : item := 0;
    uarr : item := 0;
    darr : item := 0;
  end;
repeat
  line := topline + DISP[item];
  case item of
    0 : getOPCURVEvalvel;
    1 : if doB1 then itemB1 (get_data);
    2 : if doCV1 then itemCV1 (get_data);
end;
case control of

cr : item:= item+1;
home: item:= item-1;
endk: item:= item+1;
end;
until not (item in [0..2])
or (control in menuset)
or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (DPCURVEvalvel)
  newitem;
case dowhat of
    put_box : putbox;
rub_box : rubbox;
    put_item: putitem;
rub_item: rubitem;
    get_data: getdata;
  end
end;

procedure TRANS2opcurve (dowhat: message);

const left= 5;
xref= 17;
right= 32;
top= 13;
topline= 15;
lines= 6;
bottom= 21;

var doBT : boolean;
doh2MIN: boolean;
doh2MAX: boolean;
dov2MIN: boolean;
dov2MAX: boolean;

DISP: array[0..5] of integer;

procedure SETlin;
begin
doBT := false;
doH2MIN:= true;
doH2MAX:= true;
doV2MIN:= true;
doV2MAX:= true;
end;

procedure SETxpo;
begin
doBT := true;
doH2f1IN:= true;
doH2MAX:= true;
doV2fH2:= true;
doV2MAX:= true;
end;

procedure DISPlin;
begin
DISP[0]:= 0;
DISP[1]:= 1;
DISP[2]:= 2;
DISP[3]:= 3;
DISP[4]:= 4;
end;

procedure DISPxpo;
begin
DISP[0]:= 0;
DISP[1]:= 1;
DISP[2]:= 2;
DISP[3]:= 3;
DISP[4]:= 4;
DISP[5]:= 5;
end;

procedure newitem;
begin
with worknode^ .system do begin
  case OPCODEtrans2 of
LIN: SETlin;
XPO: SETxpo;
end;
case OPCURVEtrans2 of
   LIN: DISPlin;
   XPO: DISPxpo;
end;
end;

procedure itemOPCURVEtrans2 (dowhat: message);
const field= 3;
opcurveset: scalarset= [LIN,XPO];
begin
   with worknode^.system do
      case dowhat of
      put_name: align ('opcurve= ',xref,line);
      put_data: putsymbol (OPCURVEtrans2,xref,line,field);
      get_data: thesymbol (OPCURVEtrans2,xref,line,field,
         opcurveset,control);
      end
end;

procedure itemBT (dowhat: message);
const field= 8;
   min= -10;
   max= 10;
   nullset: scalarset= [none];
   enforce= true;
begin
   with worknode^.system do
      case dowhat of
      put_name: align ('Bt= ',xref,line);
      put_data: putquantity (BT,UNITSbt,xref,line,field);
      get_data: thequantity (BT,UNITSbt,xref,line,field,
         min,max,nullset,enforce,control);
      end
end;
procedure itemH2MIN (dowhat: message);

const field= 8;
    min= 0;
    {max= H2MAX}
    lengthset: scalarset= [inches, ft, cm, m];
    enforce= true;

begin
    with worknode^.system do
    case dowhat of

        put_name: align ('H2min= ',xref,line);
        put_data: putquantity (H2MIN,UNITSh2min,xref,line,field);
        get_data: thequantity (H2MIN,UNITSh2min,xref,line,field,
            min,H2MAX/factor[UNITSh2max],
            lengthset,enforce,control);
    end
end;

procedure itemH2MAX (dowhat: message);

const field= 8;
    {min= H2MIN}
    {max= HT2}
    lengthset: scalarset= [inches, ft, cm, m];
    enforce= true;

begin
    with worknode^.system do
    case dowhat of

        put_name: align ('H2max= ',xref,line);
        put_data: putquantity (H2MAX,UNITSh2max,xref,line,field);
        get_data: thequantity (H2MAX,UNITSh2max,xref,line,field,
            H2MIN/factor[UNITSh2min],
            HT2/factor[UNITSh2l],
            lengthset,enforce,control);
    end
end;

procedure itemV2MIN (dowhat: message);

const field= 8;
    min= 0;
    {max= V2MAX}
    voltageset: scalarset= [mv, v];
enforce = true;

begin
  with worknode^.system do
  case dowhat of
    put_name: align ('V2min= ',xref,line);
    put_data: putquantity (V2MIN,UNITSv2min,xref,line,field);
    get_data: thequantity (V2MIN,UNITSv2min,xref,line,field,
      min,V2MAX/factor[UNITSv2max],
      voltageset,enforce,control);
  end
end;

procedure itemV2MAX (dowhat: message);

const field = 8;
  (min= V2MIN)
  max= 999.990;
  voltageset: scalarset= [mv,v];
  enforce= true;

begin
  with worknode^.system do
  case dowhat of
    put_name: align ('V2max= ',xref,line);
    put_data: putquantity (V2MAX,UNITSv2max,xref,line,field);
    get_data: thequantity (V2MAX,UNITSv2max,xref,line,field,
      min,V2MAX/factor[UNITSv2max],max,
      voltageset,enforce,control);
  end
end;

procedure putbox;

const y = true;
  n= false;

var item: integer;

begin
  header (put_data,top,left,right, 'TRANS 2 opcurve');
  textbox (left,top,right,bottom,n,y,y,0,0,1,1);
  line= topline;
  itemOPCURVETrans2 (put_name);
end;
procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;

var item: integer;

begin
  for item := 0 to 5 do begin
    line := topline + DISP[item];
    case item of
      0: itemOPCURVEtrans2 (put_data);
      1: if doBT then begin
          itemBT (put_name);
          itemBT (put_data);
        end;
      2: if doH2MIN then begin
          itemH2MIN (put_name);
          itemH2MIN (put_data);
        end;
      3: if doH2MAX then begin
          itemH2MAX (put_name);
          itemH2MAX (put_data);
        end;
      4: if doV2MIN then begin
          itemV2MIN (put_name);
          itemV2MIN (put_data);
        end;
      5: if doV2MAX then begin
          itemV2MAX (put_name);
          itemV2MAX (put_data);
        end;
    end;
  end;
end;
procedure rubitem;
begin
  window (xref, topline, right-1, topline+1);
gotoxy (1, 1);
circle;
  window (left+1, topline+1, right-1, bottom-1);
clrscr;
  window (1, 1, 80, 25);
end;

procedure getdata;

procedure getOPCURVETrans2;
var oldOPCURVETrans2: scalar;
begin
  with worknode^system do begin
    oldOPCURVETrans2:= OPCURVETrans2;
    itemOPCURVETrans2 (get_data);
    if (OPCURVETrans2<>oldOPCURVETrans2) then begin
      rubitem;
      newitem;
      putitem;
    end;
  end;
end;

begin
  case control of
    cr : item:= 0;
    home: item:= 5;
    endk: item:= 0;
    ccr : item:= 0;
    uarr: item:= 0;
    darr: item:= 0;
  end;
  repeat
    line:= topline+DISP[item];
    case item of
0: getOPCURVETrans2;
1: if doBT then itemBT (get_data);
2: if doH2MIN then itemH2MIN (get_data);
3: if doH2MAX then itemH2MAX (get_data);
4: if doV2MIN then itemV2MIN (get_data);
5: if doV2MAX then itemV2MAX (get_data);
end;
case control of
  cr : item:= item+1;
  home: item:= item-1;
  endk: item:= item+1;
end;
until not (item in [0..5])
  or (control in menuset)
  or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (OPCURVETrans2)
  newitem;
  case dowhat of
    put_box : putbox;
    rub_box : rubbox;
    put_item: putitem;
    rub_item: rubitem;
    get_data: getdata;
  end
end;

procedure VALVE2opcurve (dowhat: message);
const left= 37;
xref= 56;
right= 75;
top= 16;
topleft= 18;
lines= 3;
bottom= 21;

var doB2 : boolean;
doCV2: boolean;
DISP: array[0..2] of integer;
procedure SETlin;
begin
  doB2 := false;
  doCV2 := true;
end;

procedure SETxpo;
begin
  doB2 := true;
  doCV2 := true;
end;

procedure DISPlin;
begin
  DISP[0] := 0;
  DISP[2] := 1;
end;

procedure DISPxpo;
begin
  DISP[0] := 0;
  DISP[1] := 1;
  DISP[2] := 2;
end;

procedure newitem;
begin
  with worknode^.system do begin
    case OPCURVEvalve2 of
      LIN: SETlin;
      XPO: SETxpo;
    end;
  end;
end;
end;
end;
end;

procedure itemOPCURVEvalve2 (dowhat: message);

const field= 3;
optcurveset: scalarset= [LIN,XPD];

begin
  with worknode^.system do
    case dowhat of
      put_name: align ('opcurve= ',xref,line);
      put_data: putsymbol (OPCURVEvalve2,xref,line,field);
      get_data: thesymbol <OPCURVEvalve2,xref,line,field,
        opcurveset,control>;
    end
end;

procedure itemB2 (dowhat: message);

const field= 8;
  min= -10;
  max= 10;
  nullset: scalarset= [none];
  enforce= true;

begin
  with worknode^.system do
    case dowhat of
      put_name: align ('B2= ',xref,line);
      put_data: putquantity (B2,UNITSb2,xref,line,field);
      get_data: thequantity <B2,UNITSb2,xref,line,field,
        min,max,nullset,enforce,control>;
    end
end;

procedure itemCV2 (dowhat: message);

const field= 8;
  min= 0;
  max= 999.990;
flowcapset : scalarset = [{ft_sec, ft_min}];
enforce = true;

begin
  with worknode". system do
  case dowhat of
    put_name: align ('CV2=', xref, line);
    put_data: putquantity (CV2, UNITScv2, xref, line, field);
    get_data: thequantity (CV2, UNITScv2, xref, line, field,
      min, max, flowcapset, enforce, control);
  end
end;

procedure putbox;
const y = true;
n = false;

var item: integer;

begin
  header (put_data, top, left, right, 'VALVE 2 opcurve');
  textbox (left, top, right, bottom, n, y, y, 0, 0, 1, 1);
  line := topline;
  itemOPCURVEvalve2 (put_name);
end;

procedure rubbox;

begin
  window (left, top, right, bottom);
  clrscr;
  window (1, 1, 80, 25);
end;

procedure putitem;

var item: integer;

begin
  for item := 0 to 2 do begin
    line := topline + DISP[item];
    case item of

0: itemOPCURVEvalve2 (put_data);
1: if doB2
   then begin
      itemB2 (put_name);
      itemB2 (put_data);
   end;
2: if doCV2
   then begin
      itemCV2 (put_name);
      itemCV2 (put_data);
   end;
end;

procedure rubitem;
begin
   window (xref,topline,right-1,topline+1);
   gotoxy (1,1);
   clrscr;
   window (left+1,topline+1,right-1,bottom-1);
   clrscr;
   window (1,1,80,25);
end;

procedure getdata;

procedure getOPCURVEvalve2;

var oldOPCURVEvalve2: scalar;
begin
   with worknode^.system
   do begin
      oldOPCURVEvalve2:= OPCURVEvalve2;
      itemOPCURVEvalve2 (get_data);
      if (OPCURVEvalve2<>oldOPCURVEvalve2)
      then begin
         rubitem;
         newItem;
         putitem;
      end;
   end;
end;
begin
  case control of
  cr : item:= 0;
  home: item:= 2;
  endk: item:= 0;
  ccr : item:= 0;
  uarr: item:= 0;
  darr: item:= 0;
end;
repeat
  line:= topline+DISP[item];
  case item of
  0: getOPCURVEvalve2;
  1: if doB2 then itemB2 (get_data);
  2: if doCV2 then itemCV2 (get_data);
end;
  case control of
  cr : item:= item+1;
  home: item:= item-1;
  endk: item:= item+1;
end;
until not (item in [0..2])
  or (control in menuSet)
  or (control in [ccr,pgup,pgdn,uarr,darr,esc]);
end;

begin (OPCURVEvalve2)
  newitem;
  case dowhat of
  put_box : putbox;
  rub_box : rubbox;
  put_item: putitem;
  rub_item: rubitem;
  get_data: getdata;
  end
end;

begin (page3)
  control:= null;
  nodeheader (put_item);
case box of
1: nodeheader (get_data);
2: PUMPopcurve (get_data);
3: VALVE1opcurve (get_data);
4: TRANS2opcurve (get_data);
5: VALVE2opcurve (get_data);
end;

end;
case control of

cr : box := box mod 5 + 1;
home: if box = 1 then box := 5 else box := box - 1;
endk: box := box mod 5 + 1;
ccr : box := box mod 5 + 1;

eend;
until (control in \{uarr,darr,pgup,pgdn,esc\})
or (control in menuset);
if (control in \{uarr,darr\})
then begin
nodeheader (rub_item);
PUMPopcurve (rub_item);
VALVE1opcurve (rub_item);
TRANS2opcurve (rub_item);
VALVE2opcurve (rub_item);
case control of

uarr: nodeup;
darr: nodedown;

end;
end;
until (control in \{pgup,pgdn,esc\})
or (control in menuset);
PUMPcurve (rub_box);
VALVE1opcurve (rub_box);
TRANS2opcurve (rub_box);
VALVE2opcurve (rub_box);
end; (page 3)