1988

A computer-aided instructional tool for the teaching of real-time on-line statistical process control

Renford Adolphus Benito Brevett

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Applied Statistics Commons, Industrial Technology Commons, and the Other Education Commons

Recommended Citation


https://lib.dr.iastate.edu/rtd/14484

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
A computer-aided instructional tool for the teaching of real-time on-line statistical process control
by
Renford Adolphus Benito Brevett

A Thesis Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of
MASTER OF SCIENCE

Department: Industrial Education and Technology
Major: Industrial Technology

Signatures have been redacted for privacy

Iowa State University
Ames, Iowa
1988
TABLE OF CONTENTS

CHAPTER I. INTRODUCTION .................................................. 1
   Statement of the Problem .................................................. 4
   Purpose of the Study ...................................................... 5
   Need for the Study ....................................................... 5
   Assumptions of the Study ................................................ 6
   Limitations of the Study .................................................. 7
   Procedure of the Study ................................................... 7
   Definitions of Terms ..................................................... 8

CHAPTER II. LITERATURE REVIEW ........................................ 10
   Background to Statistical Process Control ............................ 10
   Future Trends in SPC .................................................... 11
   Methods of Data Analysis and Display ............................... 12
   SPC in Education ........................................................ 13
   SPC in Industries ....................................................... 14
   Process Design and Control ............................................. 15

CHAPTER III. DEVELOPMENT PROCEDURES ............................ 17
   The Construction of a Prototype Real-time System ................ 20
      Mechanical segment ............................................... 20
      Solar collector .................................................... 20
Distribution unit ............................. 20
Heat storage unit ............................. 24
The cooling system ........................... 24
House ........................................... 24
Electronic segment ............................ 24
Computer override switch ...................... 28
The power supply .............................. 28
The interface box .............................. 28
Wire connection network ...................... 34
Control board interface ....................... 34
Sensing system .................................. 34
Development of the Interfacing Software ........ 39
Flow board control ............................ 39
Input control ................................... 39
Output control .................................. 42
Development of the SPC Software for Analysis and Display .... 42
The help section ................................ 42
The statistics ................................... 42
Calibrating the sensors ......................... 43
Editor ............................................ 48
Shell to DOS .................................... 48
Exit .......................................................... 48
ROPE .......................................................... 49
The screens and display ...................................... 49
The process ..................................................... 49
Changing the variables ...................................... 50
Adjusting the system ........................................ 57
Entering and saving data .................................... 57
The user records .............................................. 60
Constructing and updating the control charts .......... 60
X-Bar and R chart ............................................ 61
CuSum chart .................................................. 61
Pareto analysis ............................................... 62
Testing of the Completed System .......................... 62

CHAPTER IV. RESULTS AND DISCUSSIONS ............... 63
Hardware ....................................................... 63
Software ....................................................... 63

CHAPTER V. SUMMARY AND CONCLUSIONS ............. 71
REFERENCES .................................................. 73
ACKNOWLEDGMENTS ....................................... 76
APPENDIX A. LISTINGS OF THE UTILITY PROGRAMS .. 77
APPENDIX B. INTERFACING SOFTWARE PROGRAM LISTINGS 139
LIST OF FIGURES

Figure 1. A schematic of the solar heat exchange system .......................... 18

Figure 2. Photograph of the solar heating system showing: house, distribution unit, storage unit, cooling system, solar panel, and computer .................................................. 21

Figure 3. A photograph of the distribution unit, showing the Three Solenoids (S1, S2, S3), the reservoir, and two pumps, P1 and P2 ................................................................. 22

Figure 4. A schematic of the sequence of the solenoids ............................ 23

Figure 5. A photograph of the heat storage unit ........................................... 25

Figure 6. A Photograph of the Electronic Segment Showing:
   Computer Override Switch (B), Power Supply (D),
   Interfacing Box (E), Wire Connection Network (F), and
   Distribution Panel (G) ................................................................. 26

Figure 7. A schematic of the electronic information flow ............................ 27

Figure 8. A schematic of the computer override switch ............................... 29

Figure 9. A schematic of the power supply ............................................... 30

Figure 10. A schematic of the interface box .............................................. 31

Figure 11. The switching circuit used for the isolating switches .................. 32

Figure 12. Artwork for circuit board of the interface box ............................ 33
Figure 14. A diagram of the distribution panel showing the cables, their location, and functions .......................... 36

Figure 15. Pinout assignments of the ribbon cables .................. 37

Figure 16. A schematic of the sensor system .......................... 38

Figure 17. Assignment of A/D channels to sensors ................... 40

Figure 18. Assignment of binary ports .................................. 41

Figure 19. Plot of data for the house sensor: (a) raw data, (b) regression line, and (c) residual .............................. 45

Figure 20. Plot of data for the panel sensor: (a) raw data, (b) regression line, and (c) residual .............................. 46

Figure 21. Plot of data for the storage sensor: (a) raw data, (b) regression line, and (c) residual .............................. 47

Figure 22. Diagram of process normal flow .............................. 51

Figure 23. A diagram of process storage flow ............................ 52

Figure 24. A diagram of process cooling system flow .................... 53

Figure 25. A diagram of process night flow ............................... 54

Figure 26. A diagram of process main down drain system ................ 55

Figure 27. A diagram of process axillary down drain System .............. 56

Figure 28. Flow diagram for the scanning process for out-of-control conditions ........................................ 58

Figure 29. Flow diagram of the adjustment procedures ................. 59
Figure 30. Flow diagram of program for analysis and display ........ 65
Figure 31. Introductory screens ........................................ 66
Figure 32. Log-on screens: (a) for professors and (b) for students ... 67
Figure 33. Main menu screen ............................................. 68
Figure 34. Sensors calibration screen .................................... 69
Figure 35. Screen of control charts ...................................... 70
CHAPTER I. INTRODUCTION

Today, in an era of constantly improving technology and increasingly competitive job markets, entry level engineers and technical teachers need to be competent and versatile in the use of computers for statistical process control (SPC). SPC is concerned with many aspects of plant design and operation. Therefore, to understand SPC one should first understand process control. According to Mollenkamp, one would not be able to come up with a short definition of process control. Some explanations that Mollenkamp suggested for the meaning of process control are:

- Keeping industrial processes at acceptable tolerance levels to maintain precision.
- Keeping industrial processes at their most efficient operating levels according to a specified standard.
- Preventing undesirable conditions in the process.
- Displaying information for plant operators’, supervisors’ and management’s decision making processes so that they can keep the industrial process running safely and efficiently (Mollenkamp 1984).

Statistical process control is the gathering of data from a process, analysis of the data by various statistical methods, and giving feedback to the process. The collection, analysis and presentation of the data can be done more quickly, efficiently and accurately using the computer than by hand.
The use of computers in process control was initiated about 30 years ago (Simmonds 1987). The first practical computer process control systems were utilized in the early 1970s (Miller 1987). Applications were restricted to centralized minicomputers which monitored a large number of processes, then recorded and displayed this information. The greatest and most significant advances have been the use of microcomputers to both collect and analyze information simultaneously, and quickly give graphical and tabular information to the operator.

The practice of process control in the industrial and technical fields is undergoing rapid technological change. The availability of inexpensive microcomputers that are powerful and capable of manipulating complex mathematical formulas make its use for sophisticated control algorithms not only possible but economically necessary. It is now evident that graduating engineers and technical teachers need to be competent in the area of statistical process control.

Hassler and Mumme (1983) at the University of Maine at Orono stated:

"Unfortunately, the supply of competent process control engineers lags far behind the demands produced by the available technology and potential applications" (p. 24).

One method of gaining such competence is from real-time practical experience in the school. According to Koppel and Sullivan, textbooks and lectures are not the best mechanisms for teaching students how to deal with real-time changes in a real plant (Koppel & Sullivan, 1986). A study done by Buxton at Teesside Polytechnic, Middlesbrough, United Kingdom (Buxton, 1985) found that
teaching programmes with a substantial proportion of practical experience on a computer system, coupled with live demonstrations of on-line computer control, stimulate considerable interest from the students, who are readily able to relate to these practical situations.

A real-time process is an actual operation that gives true information, not simulated or estimated data. If a computer is used to collect data on temperature changes within a process, analyze the data, and send feedback to the process to adjust for high or low temperatures, this is said to be a real-time process. However, if the data were generated by the computer using some simulation method, then the process is not real-time. A real-time process could be controlled by either mainframe computers, minicomputers or microcomputers.

There is now an abundance of microcomputers that are capable of collecting millions of data points and related information, analyzing the data, and recording statistical data in tabular form. By using special hardware and software configurations, microcomputers can be used to collect data and give instant and continuous real-time analysis and display of data. It is even possible to have interface devices which perform on-line control of the processes.

At present, most commercially available instructional packages that are related to process control emphasize:

1. The collection of data by simulation, historical methods, or empirical study and
2. The human transportation of data to the computer and transformation of data into graphical presentations using different software packages.

To the author’s knowledge, there is no software package for the industrial technology teacher or industrial engineer that includes a real-time statistical process control (Koppel & Sullivan, 1986; Williams & Tarrer, 1986). Consequently, this package provides a tool that can be used as instructional material for real-time statistical process control.

Statement of the Problem

Statistical process control has become a subject of more importance in recent years both in industries and in the schools. It is of importance in industries because:

(1) Industrial processes are becoming more complex than in the past due to the demand for energy conservation, demands for tighter standards, and different market demands.

(2) Industrial control strategies are taking advantage of the rapid increases in computing power and rapid decreases in computing cost, and in schools because:

(1) Industries are demanding competent engineers and industrial teachers with a broader scope of knowledge.

(2) The top schools are competing for the best students and the best jobs for them.
Therefore, it is the intent of this study to develop a low cost real-time on-line Statistical Process Control package. This package is to be used in the Department of Industrial Education and the Department of Industrial Engineering at Iowa State University as a tool for the teaching of SPC.

**Purpose of the Study**

The purpose of this study is to develop a model real-time process and software package for teaching students statistical process control by using the software to analyse the data from the model real-time process. The goal is to equip technical teachers and beginning engineers with both theoretical knowledge and practical experience in the collection, analysis and presentation of process data. This package also promotes general computer skills and an understanding of the graphical display of statistical data, and shows how they are generated and interpreted during a process. Since the students see the data, how it was processed, and how the computer responds to changes in the process, this method should aid in the development of the students' analytical skills.

**Need for the Study**

Statistical process control is a difficult subject to teach because it ranges from statistical analysis using abstract ideas such as calculations in the mathematical formulas, to realities such as analysis of how the weight of a product is affected by the speed of the process. Although SPC is applicable to many areas of engineering, presently, most SPC packages are being implemented in the chemical engineering
area. Chemical engineering programs in several universities (Koppel & Sullivan, 1986; Karim, 1984; Luecke & Lin, 1986) have developed or utilized commercial SPC programs for instructional purposes. These programs predominantly utilize expensive software/hardware systems such as the IBM ACS (Koppel & Sullivan, 1986). To the author’s knowledge, no low cost (less than $500) software/hardware packages for instructional purposes exist in the industrial technology areas. The more costly packages that are available require more extensive knowledge in statistics, mathematics and industrial processes than students in the target audience of this project possess.

There is a growing need for knowledge about SPC in industry (Koppel & Sullivan, 1986; Hassler & Mumme, 1983). The students for which this activity is intended should be exposed to this area since most will enter the industrial labor market in which these skills will be utilized.

Assumptions of the Study

The following assumptions were made concerning the study:

1. The components used were stable and the values were as stated by the manufacturer.
2. The computer used to develop the software was a true IBM compatible machine.
3. The IBM interfacing board used was a true IEEE 488 standard board.
4. Any errors introduced due to speed of the board or the computer are negligible.

5. The areas of SPC dealt with in this package sufficiently cover the needs for the level of the intended audience.

Limitations of the Study

1. The software was designed to run only on IBM compatible microcomputers.

2. The process used was a model of an industrial process.

3. The software was customized to run with the IBM interfacing board and may not run with other interfacing boards.

Procedure of the Study

This study was conducted in the Department of Industrial Education at Iowa State University. A model of a controlled solar heating process was built. The computer was interfaced to this process. The software was developed to control and analyze the process using SPC techniques. The software developed has the capability to perform:

1. Storing and retrieving of data.

2. Control chart analysis ($\bar{X}$, R, and CuSum).

3. Pareto analysis.

4. Calculating and Plotting of various statistics (mean, range, etc.).
5. Updating the status of all instruments and machines.

6. Editing of any text files.

The computer, with the aid of the interface board, receives input from the process by various sensing mechanisms. The IEEE 488 standard board, developed by International Business Machines (IBM), is the interface between the computer and the process. The software package, in combination with the PC and the interfacing equipment was used to control and analyse the process showing:

1. Input and output of data.
2. The storage of data to disk or any other storage mechanism.
3. The sensing of the temperature inside and outside the system.
4. The opening and closing of the solenoids to regulate water flow.
5. Switching pumps ON and OFF.
6. Plotting of the data.

**Definitions of Terms**

The following definitions were adopted for the purpose of this study:

1. **IEEE 488 standard**: A convention of recommended codes developed by the Institute of Electrical and Electronics Engineers (IEEE). It is a digital interface system for programmable measuring instruments.

2. **DACA**: The IBM PC Data Acquisition and Control Adapter. It provides the capability to control and monitor processes within a sensor based system.
3. GPIBA: The General Purpose Interface Bus Adapter. It allows access to and control of instruments and devices that are compatible with the IEEE 488 standard.
CHAPTER II. LITERATURE REVIEW

Background to Statistical Process Control

Computers were tentatively used for the control of machines about 30 years ago (Simmonds 1987). According to Simmonds, fears of reliability restricted applications until more powerful and reliable machines based on integrated circuits were introduced. Computers were then fully used for control functions. The rapid advancement in digital-electronics technology accelerated the use of computers in industrial process-control. This rapid advancement led to the smaller, cheaper, more powerful and more reliable microcomputers. Several methods for the collection of useful analytical data, the reporting of results and subsequent feedback to the process using these microcomputers were tried. Data were collected manually and analyzed using statistical theory such as Shewhart control charts and acceptance sampling plans. Better methods that would ease the manual aspects of the process have been sought ever since. Hunter (1987a) puts this in perspective when he remarked:

Remarkably, if a new piece of hardware is invented, one can expect to see its prompt arrival within industry. But what about a new statistical tool, something to help an individual interpret data, or create information-laden data? (p. 94).

The statistical tool that yields the maximum amount of information from the minimum amount of data is SPC. SPC can be used to measure, analyze and
characterize any process (Gupta, Macry, Pena, & Westerman 1987). They define SPC as:

... the use of statistics to determine if a process is behaving naturally (only variation that is inherent in the process), to detect any unnatural behavior, and to determine and eliminate the causes of unnatural behavior (p. 23).

Because the various methods of detecting variation in a process require very sophisticated instrumentation and advanced mathematical formulas, many factors are considered before the implementation of SPC. In order to implement SPC in a process it is important to know if the effect it will have on the process will be immediate or delayed.

**Future Trends in SPC**

Although SPC is new to some industries the concept has been around for a long time and will continue to be a part of the quality improvement technique. Experimental designs such as that developed by Box in 1957, Ott in 1975 and Taguchi in 1986 (Hunter 1987a) are all stepping-stones to SPC. According to Hunter (1987a), for future success, workers and managers should be statistically literate so that they can communicate quantitative information succinctly. Marilyn Hart and Robert Hart (1987) at the University of Wisconsin-Oshkosh state:

The evolution that has occurred in going from "quality control" to the "total control of quality" has, for its foundation, a new recognition of the role played by statistics-the inspection data-obtained from the process (p. 96).
The next step in SPC seems to be automation. Hubele and Keats (1987) at Arizona State University note that SPC must adapt to keep pace with current technology. They further note that automation can be implemented by the use of data acquisition systems (DAS) along with the complex data-gathering processes that are already in place (Hubele & Keats, 1987). As there are various sophisticated methods used for gathering the data there are also various sophisticated methods used for analyzing the data and for displaying the results.

**Methods of Data Analysis and Display**

The major method of analyzing and displaying data is the use of control charts. The Shewhart control chart was the only major charting technique used in industry in the 1960s (Hunter 1987a). This charting technique, Hunter said, could be used not only to detect variation in the mean but could include variables data, attribute data or simple counts, and other variants - for example, median, run, first difference and moving range.

Another method of analyzing and displaying data is to use exponential weighted moving average (EWMA) for the construction of control charts for the mean. Sweet (1986) suggested that EWMA charting for the mean and the standard deviation or the variance of a process should be used for process monitoring and not for process capability analysis.

Another method of analyzing and displaying of data is the use of the zone control chart. Jaehn (1987) described the zone control chart as a maximum
simplification of the Shewhart chart. This chart has eight zones with scores assigned to each zone. Jaehn further states that zone charting takes into account a shift in the process while the process is still within limits. One disadvantage to this method, he noted, is the lack of early warning to signal when a change in level occurs.

A final method of analyzing and displaying data is the use of time series analysis. According to Hubele and Keats (1987), time series analysis is well suited for situations where observations collected from a process are correlated and non-normal, such as a DAS. Hubele and Keats categorized time series analysis into: Historical - relying on past data, Transfer function models, and Bayesian inference models.

**SPC in Education**

Many universities now recognize the need to teach process control even at the undergraduate level. Hassler and Mumme (1983) at the University of Maine at Orono, developed a course sequence in process control in the chemical engineering department at the undergraduate level. The sequence includes process dynamics and control, digital process control, and advanced process control. Hassler and Mumme recognized that the conventional chemical engineering curriculum does not provide graduates with a working knowledge of "modern" or multivariable control. They believed that the new program is unique and valuable. However, they found that it is necessary for the following to be included in the program: mathematics, instrumentation hardware, computer hardware and software (on-line, real time
system), interfacing of computer and process, computer operating system maintenance, industrial experience, and the areas of process control covered by the courses.

The chemical engineering faculty at Auburn University also recognized the need for trained engineers in process control and therefore revised their undergraduate curriculum in 1984 (Williams & Tarrer, 1986). The general observation by the faculty was that students are better prepared in process control than under the previous curriculum.

Computers are becoming an integral part of the educational program. Both mainframe computers and microcomputers are being widely used in several universities for the teaching of process control. The University of Waterloo and Purdue University contracted with IBM to use IBM’s Advanced Control System (ACS) to develop process control courses (Koppel & Sullivan, 1986). The ACS system is a licensed program providing a framework for implementing plant wide process control. Both schools used the IBM 4341 mainframe computer (Koppel & Sullivan, 1986). Self-study guides were written at Purdue and tested at both Purdue University and the University of Waterloo.

**SPC in Industries**

Most US manufacturers and entrepreneurs, faced with corporate philosophy and budget allocations, are also faced with the need for the highest quality product or service. SPC training has become as important and expensive as its
implementation. Bajaria (1987) said that managers are wrong to think that typical training sessions prepare their staff for executing SPC. Bajaria further notes that most SPC training focuses on philosophy and mathematical details and not on execution strategy or skills.

**Process Design and Control**

A real-time process design differs greatly from other data processing designs. Real-time systems must be able to meet both external and internal deadlines. Various tasks must be coordinated asynchronously and concurrently. These tasks are interfaced with a process external to the computer. An appropriate design for such operations requires extensive planning and testing. The process design must therefore include real-time data-flow diagrams (Williams, 1987), charting and display, and process modeling framework (Jones, 1987). The data-flow diagrams depict the concurrent tasks of the system and the interfaces between the tasks. The process design should include the method of control for the process.

There are many different methods of process control. Roat and Melsheimer (1987) at Clemson University observed the frequent use of the Ziegler-Nichols first-order plus deadtime (FOPDT) model for overdamped systems. The system under control can either be open or closed loop. Adb-El-Bary’s (1983) digital control liquid level experiment and Famularo’s (1987) computer-controlled heat exchange experiment, illustrate the use of various control modes. These modes
include proportional (P), proportional-integral (P-I), proportional-derivative (P-D), and proportional-integral-derivative (PID). The PID mode controller is the ultimate in control of continuous process. This project implements the PID mode of control.
CHAPTER III. DEVELOPMENT PROCEDURES

The development of this package was divided into four major areas: the construction of a prototype real time system, the development of the software to interface the computer to the system, the development of the SPC software for analysis and display of data, and the testing of the completed package.

The system chosen for this project is a solar heat exchange unit (see Figure 1). Solenoids are used to open or close valves to regulate water flow, and radiators exchange heat with the surrounding medium. A tank is used as a storage for excess heat. Pump 1 is used to circulate the water through the system. Pump 2 is used to circulate water through the radiators when the light source and controlling temperature are low. The parameter to be controlled in this project is the ambient air temperature of the house.

The software for analysis and display was developed using the Turbo Pascal programming language. The software has the following capabilities:

- Input of data on-line or from disk
- Storage of data on disk
- Data analysis using various statistical methods
- Display of data on screen and output to printer or other devices
- Control of process through feedback
Figure 1. A schematic of the solar heat exchange system
The software for the interface between the computer and the system was also developed using the Turbo Pascal programming language. The IEEE 488 standard board was used to obtain sensor reading and to control the solenoids and pumps. The program for the calibration of the instruments was designed first, followed by the program for reading the sensors and controlling the solenoids and pumps. The program used three A/D converters and eight bits of binary input/output. The A/D converters and the I/O’s are on the IEEE 488 standard board.

The system was tested by Dr. Miller, Dr. Even, Dr. Wolansky. The testing of the system included:

- Checking for the ability to input and store data
- Checking the consistency, display, and clarity of information or error messages
- Checking the control and responses of the process to variations in the system
- Checking the accuracy and validity of results.

The testing stage was the final step in the development of this package. A copy of the software and documentation can be obtained from the Industrial Education Department at Iowa State University.
The Construction of a Prototype Real-time System

This system is a solar heating unit (see Figure 1&2). It can be divided into two segments, mechanical and electrical.

**Mechanical segment**

This segment consists of: solar collector, distribution unit, storage unit, cooling system, and house. The construction of each unit and the part each plays in the process is discussed below.

**Solar collector**

The solar collector panel used in this project is 20 inches long, 18 inches wide, and 4 inches deep. Its construction is very simple and inexpensive. The walls and back plate are built from aluminum, and the front is covered with plate glass. Copper pipe tubing circles through the panel. For maximum heat capture copper flanges are glued to the pipe to give increased surface area, and the inside of the collector is painted black. The panel has insulation to prevent heat loss.

**Distribution unit**

The distribution unit consists of three switches (solenoids were used), two pumps, and a reservoir (see Figure 3). The switches can select either one of two inlets and channel flow to a single outlet. An alternative selection of switches is a single inlet and channel output to one of two outlets. These solenoids are electronic switchable using 12 to 18 volts DC, and are used in the sequence shown in Figure 4.
Figure 2. Photograph of the solar heating system showing: house, distribution unit, storage unit, cooling system, solar panel, and computer
Figure 3. A photograph of the distribution unit, showing the Three Solenoids (S₁, S₂, S₃), the reservoir, and two pumps, P₁ and P₂
Figure 4. A schematic of the sequence of the solenoids
All three solenoids were mounted on a piece of board along with the reservoir (a one quart plastic container) and the two pumps. The board was mounted vertically near the work station. The two pumps (taken from a car) are also operated off a 12 to 18 volts unregulated DC power supply. These pumps are excellent for automatic flow control.

**Heat storage unit**

The heat storage unit, shown in Figure 5, consists of an enclosed insulated container (36 quart Igloo picnic cooler) (blue, see Figure 2) with a heat exchanger, some rocks and the heat transfer liquid inside. The liquid used was ethylene glycol based car antifreeze.

**The cooling system**

The cooling of the heat transfer liquid is effected by a simple water and air flow cooling mechanism. The liquid flows through a spiral copper tubing submerged in icy water (in the red picnic cooler of Figure 2) and a straight copper tubing that is fan cooled.

**House**

The prototype house used in this project is a child’s play barn, (about 1 cubic foot); the size of the house was small compared to the size of the solar panel (see Figure 2). It allows easily detected temperature variations.

**Electronic segment**

This segment consists of: computer override switch, power supply, interface box, wire connection network, control board interface, and sensing system (see Figure 6). The interactions between all of the electronic parts are shown in Figure 7.
Figure 5. A photograph of the heat storage unit
Figure 6. A Photograph of the Electronic Segment Showing:

Computer Override Switch (B), Power Supply (D),
Interfacing Box (E), Wire Connection Network (F), and
Distribution Panel (G)
Figure 7. A schematic of the electronic information flow
**Computer override switch**  The computer override switch uses solid-state circuitry to allow the computer to turn on/off power to the system. This switch can also be used as an emergency switch to shut off all power to the system. As shown in Figure 8 the circuit used an analog single pole double throw (SPDT) switch with center off and a solid-state power relay (opto isolated). This solid-state power relay is single pole single throw (SPST) and is normally open and incorporates inverse-parallel SCR output devices. The high input to output isolation makes these relays high in noise immunity to the drive circuit.

**The power supply.**  The power supply used has both regulated and unregulated outputs. The regulated output was used for the interface box which requires regulated 5VDC, and the unregulated output was used for the pumps and solenoids. The solenoids require more current than can be supplied by a regular 12VDC transistor regulator, so, using an unregulated supply will eliminate the purchase of a more expensive regulator. Since the pumps have the same requirements, the same method was used for them.

The power supply uses three separate transformers; one for each pump and one for the solenoids and the 5VDC regulated circuit (see Figure 9).

**The interface box**  The interface box holds the isolating and switching circuits for the system. Signals received from the control board are buffered, amplified and used to activate relays that in turn switch a solenoid or pump. This
Figure 8. A schematic of the computer override switch
Figure 9. A schematic of the power supply
Figure 10. A schematic of the interface box
Figure 11. The switching circuit used for the isolating switches
Figure 12. Artwork for circuit board of the interface box
box uses six isolating switches, six relays and a 8055 buffered signal amplified chip (ECG 2021, a 8 Digit/Segment Darlington Array/Driver) (see Figures 10,11,&12).

**Wire connection network** The wire connection network consists of the three ribbon cables from the distribution panel, the ribbon cable to the interfacing box, and the wires to the sensing system. Figure 13 shows how these cables and wires are connected.

**Control board interface** The control board interface has three sections: Data Acquisition Control Adapter (DACA), distribution panel, and three interface cables.

The DACA board (IBM) was installed as received into one of the expansion slots in the PC. The board comes with an IEEE 488 standard port. A flat cable plugged into this port connects the distribution panel to the PC.

The distribution panel has 60 connectors for external access to the DACA. For easy access, three ribbon cables with a 16 pin plug terminal were installed (see Figure 14); they have the pinout shown below in Figure 15. Only eight of the sixteen binary input/output bits (B0 - B7) were needed for this project.

**Sensing system** A photo-resistor and three thermistors were used as sensors. A voltage divider circuit was used to measure the voltage drop across the sensor due to change in light intensity (in the case of the photo-resistor) or change in temperature (in the case of the temperature transducer). The circuit used is shown in Figure 16. The A/D converter on the DACA board is used to measure the voltage
Figure 13. A schematic of the wire connection network
Figure 14. A diagram of the distribution panel showing the cables, their location, and functions.
Figure 15. Pinout assignments of the ribbon cables
Figure 16. A schematic of the sensor system
(see Figure 15 for the pinout of the DACA cables). Figure 17 shows which A/D channels were used for each sensor.

**Development of the Interfacing Software**

The interfacing software was written in Turbo Pascal version 4.0 (Borland International). The purpose of this software is to control the pumps, the solenoids and the main system switch. The software also reads the A/D ports and writes information to the D/A ports. The procedure and functions for this software are in SPC_Util (see Appendix A). This software has three major categories of control: Flow board control, Input control, and Output control.

**Flow board control**

Switching the solenoids directs the flow of liquid through the system. The three lower bits (B0 - B2) of the binary output of the DACA board were used to control the solenoids. The fourth bit was used to control the down-drain solenoid value. Bits 5 and 6 turn the pumps on and off (see Figure 18).

**Input control**

The binary inputs were shorted with the corresponding binary outputs so that the program can detect the status of each bit. The four sensors were hooked up to the A/D ports of the DACA.
Figure 17. Assignment of A/D channels to sensors

- A/D0: Panel Photoresistor
- A/D1: Panel Thermistor
- A/D2: House Thermistor
- A/D3: Storage Thermistor
- D/A0: Used for Switching
Figure 18. Assignment of binary ports
Output control

Only one of the D/A ports was used for this project. This port was used to send 5V to the TTL circuits of the interfacing box.

Development of the SPC Software for Analysis and Display

The software used for SPC analysis is called ROPE (Real-time On-line Processing Environment). There are seven sections: Help, Statistics, Calibration, Editor, Shell to DOS, Exit, and ROPE. Each section is accessible from a main menu by using the function keys [F1 - F6, F10]. The user manual for ROPE is in Appendix E.

The help section

On-line help is available from anywhere within the program. This is activated by pressing the F1 key. The unit SPC_Help (see Appendix A) has the routines to access the help file and display information.

The statistics

Simply pressing the F2 key allows the user to do various statistical analysis of data collected from the system as well as any other data. Data can be entered at the keyboard as well as from an existing data file. The units for this segment (SPC_Kal and SPC_Cal) are in Appendix C.
Calibrating the sensors

Four sensing devices were used for sensing temperature and light intensity. Thermistors were used to sense the air temperature in the house, the liquid in the solar panel, and the liquid in the storage tank. A photo-resistor was used to sense the light intensity on the panel. These sensing devices were calibrated individually for the project. However, the software allows for recalibration of these devices at any time by pressing the F3 key at the ROPE main menu. The calibration procedure consists of three steps:

Step 1. Setting up the sensors and thermometers.
Step 2. Collecting and recording data using ROPE.
Step 3. Calculation of the regression lines using ROPE.

Step 1. A thermometer is placed next to each thermistor, oriented for convenient reading. The system is started as usual and allowed to run for 15 minutes, which allows it to stabilize. Dry ice is then placed in the house to lower the temperature. Data are collected after stabilization at the lowest temperature of about 48 F.

Step 2. Readings are taken at 2 degrees F intervals and the corresponding voltages recorded. The computer automatically records the latest-read voltage when the user inputs the temperature. The data collected for each sensor are in Appendix D.
Step 3. The data collected in step 2 are used to calculate the regression line. A four-predictor multi-regression routine is used for this calculation. The regression equation used is of the form: 

\[ Y = ax + bx^2 + cx^3 + dx^4 + K \]

Where:

- \( Y \) = Temperature predicted
- \( a \) = Predictor 1
- \( b \) = Predictor 2
- \( c \) = Predictor 3
- \( d \) = Predictor 4
- \( K \) = Constant (Y-intercept)
- \( x \) = Voltage

Plots of the data collected, the regression lines, and the residuals of the predicted values are presented in Figures 19 thru 21. The regression equations found are:

House sensor

\[ Y = -73.7381x - 7.07285x^2 + 10.16346x^3 - 1.77828x^4 + 227.2725 \]

Panel Sensor

\[ Y = 78.43279x - 62.3795x^2 + 15.08995x^3 - 1.45083x^4 + 103.5134 \]

Storage sensor

\[ Y = 302.9336x - 254.934x^2 + 77.83091x^3 - 8.26061x^4 + 24.77183 \]

The software is in SPC_Kal (see Appendix C).
Figure 19. Plot of data for the house sensor: (a) raw data, (b) regression line, and (c) residual
Figure 20. Plot of data for the panel sensor: (a) raw data, (b) regression line, and (c) residual
Figure 21. Plot of data for the storage sensor: (a) raw data, (b) regression line, and (c) residual
Editor

The editor, Microstar, was adapted from the Turbo Pascal Editor Toolbox (Borland International). It allows for the editing of both text and data files. Information on using this editor is in Appendix E. The complete editor package can be obtained from Borland International. In this project, it is run by the software ROPEMs (see Appendix A), and activated by the F4 key.

Shell to DOS

Users can temporarily go to DOS from the ROPE environment by using this capability. There are two levels of shelling to DOS. Level 1 allows full access to DOS but requires the user to enter ROPE to return to the ROPE environment. This mode is activated by the F5 key. Level 2 (activated by pressing the RETURN key while in Level 1) requires the user to enter EXIT at the DOS prompt to return to Level 1 and the ROPE environment. At Level 2, ROPE is invisible to the user. One does not exit the program using this routine, therefore, memory is still in use and any large program may not be executed in this mode. The codes for this feature are in the SPC_Menu (see Appendix A).

Exit

At the end of each session, this routine resets the system for the next user. It is possible to exit to DOS or the program using this routine but this is allowed only for users in the unprotected mode or user with a status of professor. If exit enable is
set then this routine will exit the program gracefully. The unit SPC_Menu (see Appendix A) contains the codes for this feature.

**ROPE**

By pressing the F10 key, the routines for the analysis and graphical presentation of data are accessed. This function includes the following: screens and display, process, changing variables, adjusting the system, and entering and saving data. The routines are in SPC_Grap (see Appendix C).

**The screens and display**

The display of information on screen uses routines from Turbo Pascal Editor and Graphics Toolboxes. This program makes use of the multi-window techniques and the graphics display of Turbo Pascal. The graphics screen shows plots of: X-bar chart, R chart, CuSum chart, Pareto analysis, scatter plot of the raw data, and information about the state of the process. Routines for screen management are found in the units SPC_Util and Introd (see Appendix A).

**The process**

The actual SPC process routines are in the unit SPC_Proc (see Appendix B). The main goal of the process is to maintain a temperature of 85 degrees F. The duration of the process is based on the cycle time set by the user or the internal default. The system continuously cycles through three phases: normal flow, storage flow and cooling system flow. Normal flow is the flow of liquid through the panel
and house (see Figure 22). Storage flow is the flow of liquid through the storage unit (see Figure 23). Cooling system flow is the flow of the liquid through the cooling system (see Figure 24) to cool the panel when a high critical temperature is reached on the panel and storage of heat energy is not required. Night flow is the flow of liquid between the storage unit and the house to extract heat from storage (see Figure 25). The house temperature is measured, plotted and stored at equal intervals set by the user. The light intensity is also supervised, so when the light intensity is low and there is energy in storage, night flow is used to heat the house. Figures 22 thru 27 below show the various types of flow. The system can be down drain for maintenance or in emergency situation. The main down drain flow (see Figure 26) drains the main system and the axillary down drain flow (see Figure 27) drains the other sections as shown.

**Changing the variables**

The behavior of the system depends on: sampling time, cycle time, and control limits. All the default values for these variables are in SPC_Id (see Appendix A), and they can be changed at the beginning of or during the process. The maximum number of samples for the process is 100. This maximum cannot be changed by the user.
Figure 22. Diagram of process normal flow
Figure 23. A diagram of process storage flow
Figure 24. A diagram of process cooling system flow
Figure 25. A diagram of process night flow
Figure 26. A diagram of process main down drain system
Figure 27. A diagram of process axillary down drain System
Adjusting the system

The system is adjusted automatically when out-of-control situations occur, or when trends indicate that the system is nearing an out-of-control situation. The adjustment includes changing the duration for each phase within a cycle. Figures 28 and 29 show how the process is scanned for out-of-control conditions and how it is adjusted when such conditions are detected. The routines in SPC_Grap (see Appendix C) control these adjustments.

Entering and saving data

Data for analysis may be entered from the keyboard, the disk drives or in-line from the process. The format of the data depends on how the data will be analyzed. The codes for this are in SPC_Cal (see Appendix C). Data to do basic statistical analysis are stored in a text file in the following format:

First line has the number of data points to analyze.

All other lines have five sets of data:

[Y-value x1-value x2-value x3-value x4-value]

The user may enter these values from the keyboard if desired.

Data to be analyzed for the SPC process have two formats.

1. Data for the calibration routines need to be in the following format:

The first 12 lines may be used for instruction or other labeling.

All other lines have four sets of data:
Figure 28. Flow diagram for the scanning process for out-of-control conditions
Figure 29. Flow diagram of the adjustment procedures
[Device-calibrating Temperature Voltage Time Elapsed].

2. Data for the control charts are in the format:

[Data # Temperature Time].

This program also maintains information about the users. This information can be entered by the professor or any user in the unprotected mode.

**The user records**

There are 2 modes (protected and unprotected) and 3 statuses (professor, graduate, and student) that apply to users of ROPE (see the user manual in Appendix E). An unprotected mode allows for access to more sensitive information and changes to the system than a protected mode of the same status. A protected mode student may only change their own passwords and see their own record. The unprotected professor may do any of the following for any user: change passwords, change grades, and -add, delete, or make new list of- users. The user records are information entered by the professor or the user in the unprotected mode. Each user information is entered and stored in a file called ROPE.LST. ROPE.LST is a file of records with thirteen fields which can be updated by the professor or any authorized user in the unprotected mode. The codes are in SPC_Rec in Appendix D.

**Constructing and updating the control charts**

The three control charts used in the project are X-bar, R and CuSum. These charts were constructed using routines from the Turbo Pascal Graphics Toolbox.
**X-Bar and R chart**

The units SPC_Grap (see Appendix C) contains the routines used to plot the charts. Each chart is plotted when the screen is updated. Some of the graphics routines are in the Turbo Pascal Toolbox. The technique used is the U.S. standard of the plotting of the control charts. This technique differs from the British in the control limits used. British limits are:

- **Warning limits** = \( \bar{X} \pm 1.96 \sigma / \sqrt{n} \)
- **Action limits** = \( \bar{X} \pm 3.09 \sigma / \sqrt{n} \)
- American limits are: \( \bar{X} \pm 3 \sigma \)

The British have a tighter tolerance limit for both warning and action limit.

**CuSum chart**

Routines for the plotting the CuSum chart are also found in SPC_Grap in Appendix C. The reference value used is the desired temperature of the process which is set by the operator (user). The equation is of the form:

\[
S_r = \sum_{i=1}^{r} (X_i - K) = S_{r-1} + (X_r - K)
\]

Where:

- \( S_r \) = The Cumulative Sum (CuSum)
- \( K \) = Reference Temperature
- \( X_i \) = Sample temperature value
Pareto analysis

The routines for the pareto analysis are in the unit SPC_Grap (Appendix C). The identifying of cost and quality can be detected by using this type of analysis. By applying a reasonable cost to each method of adjustment and each type of flow, an analysis can be done to detect quality and product capability.

Testing of the Completed System

The process tested by the committee that oversee this project. They are from the Industrial Education department and the Industrial engineering department. The following three steps guided them through the test.

Step1. Each was given a passwords to log-on to the system. They entered the given information in five different ways until they were able to log-on. They also changed passwords make a student list and edit (add, delete and change information) the list.

Step2. They ran the system using the editor, shell capability and help sections. They were given instructions that included errors and invalid information. This was to test the error handling and invalid data detection capability of the system.

Step3. They then entered ROPE where they could run the process and change variables. They were given instructions on what to look and test for. They were also allowed to calibrate the sensors and to collect and analyze data from the process.
CHAPTER IV. RESULTS AND DISCUSSIONS

The development of this project can be divided into two distinct areas: Hardware and Software. Since the project was primarily a teaching tool only concepts that are considered pertinent to a teaching environment were emphasized. This section will discuss the hardware aspect first and then the software.

Hardware

The hardware includes the complete solar heating system, the electrical circuitry, and the computer. The solar heating system was constructed and tested for proper operation and interface with the rest of the system. The one thing to note about the solar system is the use of the appropriate pump. The pumps should be able to run constantly without becoming over heated. They should be able to operate on a 12 to 18 volts DC power supply. The power supply shown in Figure 6 of Chapter 3 could be modified to include a full bridge rectifier and a different regulating circuitry that can handle high current. This will enable the use of regulated outputs to the pumps and solenoids.

Software

The software includes programs for controlling the operations of the solar system, for analyzing and displaying information about the process, and a user management program. The controlling program used routines that may be specific to the IBM DACA board used for this project. Included in Appendix B are routines
for the controlling program, and for various operations of the process, including the
types of flows and the switching of the devices. Appendix F contains other technical
and programming information about the DACA board.

The flow diagram in Figure 30 shows the program for analyzing and
displaying information about the process. Figures 31 to 35 show some of the screens
that were generated by this program. In generating the control charts the user is
allowed to change various parameters such as the control limits, the process cycle
time, and the sampling rate, to see what effect such changes have on the process.
Figure 35 shows the charts and the temperature in the house, on the panel and in the
storage. The system will respond to changes to the process cycle time. The effects of
changing the sampling rate can be easily noted by the behavior of the system in
terms of frequency of adjustment to the process. The user management program is
used to setup files and initialize variables that cannot be changed by an end user that
is in a protected mode. The protected mode specific to this program is one of such
variables that can be assigned to any user. Protected mode used here should not be
confused with the protected mode defined by the DOS operating system on an AT,
286 or 386 machine. It is recommended that only the professor be in the unprotected
mode. This will encourage privacy and discourage piracy, pilfering, cheating, and
misuse of the system.
Figure 30. Flow diagram of program for analysis and display
Welcome to ROPE (Real-Time On-Line Processing Environment) at
Iowa State University Ames Iowa. This package was designed by
Menard Brevett as part of his masters thesis. It was conducted
under the guidance of his major professor William G. Miller and
also Professor John Keen Jr. of the Industrial Engineering
Department and Professor William Walensky of the Industrial
Education and Technology Department.

--> Press any key to continue <--

This package is designed as a tool for a real-time on-line
computer-based demonstration of Statistical Process Control (SPC).
This package is designed to be used by, but not limited to
undergraduate Industrial Technology, Industrial Education
and Industrial Engineering students at Iowa State University.
It is easy to understand the concepts of SPC that are discussed
and illustrated in this package, since the knowledge of only
introductory statistics is assumed.

--> Press any key to continue <--

This program is menu driven with consistency in the window
display. Since computer-based instruction is unique, and
computerization has proliferated in the technical world, the
opportunity to gain close acquaintance with this area will be
advantageous.

--> Press any key to continue <--

Questions or comments can be addressed to:

W. G. Miller
Industrial Education and Technology
Iowa State University
Ames, Iowa 50011.
Ph. (515) 515-6944

--> Press any key to continue <--

Figure 31. Introductory screens
The following Information Must be provided to Run ROPE

Name : William G. Miller
Course : Iedt 615
Times Used : 1
Status : Professor
Mode : Unprotected
Last Used:
Date : 05/13/88
Time : 01:33:am
Progress : First Time User
Password : *****
G.P.A. : 4.00 Grade <A>

F1-Make Student List
F2-See List
F3-Edit list
F7-Change Password
F9-Continue...

The following Information Must be provided to Run ROPE

Name : Renford Brevett
Course : Iedt 615
Times Used : 2
Status : Student
Mode : Protected
Last Used:
Date : 7/20/1988
Time : 12:40:55am
Progress : Good
Password : *****
G.P.A. : 3.95 Grade <A>

F1 Help/F7 Change Password / Any other Key To Cont...

Figure 32. Log-on screens: (a) for professors and (b) for students
Figure 33. Main menu screen
Figure 34. Sensors calibration screen
Figure 35. Screen of control charts
CHAPTER V. SUMMARY AND CONCLUSIONS

The intent of this study is to develop a model low cost real-time on-line process and SPC software package for teaching students statistical process control by using the software to analyse the data from the model real-time process. The goal is to equip technical teachers and beginning engineers with both theoretical knowledge and practical experience in the collection, analysis and presentation of process data. The students for which this activity is intended should be exposed to this area since most will enter the industrial labor market in which these skills will be utilized. This package also promotes general computer skills and gives an understanding of the graphical display of statistical data, and how they are generated and interpreted during a process. Since the students see the data, how it was processed, and how the computer responds to changes in the process, this method should aid in the development of the students' analytical skills. This package is to be used in the Department of Industrial Education and the Department of Industrial Engineering at Iowa State University as a tool for the teaching of SPC.

Statistical process control is the gathering of data from a process, analysis of the data by various statistical methods, and giving feedback to the process. The collection, analysis and presentation of data using an IBM compatible computer, was done more quickly, efficiently and accurately than by hand. Statistical process control is a difficult subject to teach because it ranges from statistical analysis using
abstract ideas such as calculations in the mathematical formulas, to realities such as analysis of how the weight of a product is affected by the speed of the process.

By using special hardware and software configurations, microcomputers can be used to collect data and give instant and continuous real-time analysis and display of data. It is even possible to have interface devices which perform on-line control of the processes. In this case, the microcomputer was used to collect data on temperature changes within a process, analyze the data, and send feedback to the process to adjust for high or low temperatures, which is called a real-time process.

A model of a controlled solar heating process was built. The computer was interfaced to this process. The computer, with the aid of the interface board, receives input from the process by various sensing mechanisms. An IEEE 488 standard board, developed by International Business Machines (IBM), is the interface between the computer and the process. The software package, in combination with the PC and the interfacing equipment was used to control and analyse the process showing: input and output data, the sensing of the temperature inside and outside the system, and plotting of the data. Software was also developed to control and analyze the process using SPC techniques. The software developed has the capability to perform: storing and retrieving of data, control chart analysis ($\bar{X}$, R, and CuSum), Pareto analysis, calculating and plotting of various statistics (mean, range, etc.), updating the status of all instruments and machines, and editing of any text and data files.
REFERENCES


ACKNOWLEDGMENTS

The author would like to thank Dr. W. G. Miller for his help and guidance throughout this project, and also for his great enthusiasm and interest in a common field which were the main catalysts behind this project. Thanks go also to Karen Myers who helped in the collection of the data.

Most thanks go to a most devoted and tolerant friend Dr. Carol A. Simmons, whose help goes far beyond descriptions in this domain.
APPENDIX A. LISTINGS OF THE UTILITY PROGRAMS
Unit SPC_ID;

interface

Uses
  Crt.
  Turbo3.
  GDiver;

Const
  Windows = 10;
  ESC = #27;
  RETURN = #13;

type
  Str80 = string[80];
  Str11 = string[11];
  Str3  = String[3];
  Str64 = String[64];
  Str20 = String[20];
  Str14 = String[14];
  lDtype = (Analog,Binary);

UserDataType = Record
  Date     :Str11;
  Time     :Str11;
  Password :Str14;
  FirstName:Str20;
  LastName :Str20;
  Course   :Str11;
  SSNumb     :Str11;
  Progress  :Str80;
  Grade    :String[1];
  GradePoint:Real;
  Status   :Str11;
  Mode     :Str11;
  TimesUsed:Integer;
End;

DataFile = Text:
RecordFile = File;
TableData = Array [1..12] of real;

TextChars = Array [1..3] of Byte;
TextCharArray = Array [32..128] of TextChars;
WtabArray = Array [1..Twindows,1..5] of Integer;
    (X0,Y0,X1,Y1,LineNo)

SampleDataType = Array [1..100] of real;

Const
SelectHighOrderByte = $0001;  { These constants are the hexa-
AnalogReadControlRegisterOffset = $0000;  { decimal offsets that must be }
AnalogReadRegisterOffset = $2000;  { added to the CardBaseAddress }
AnalogStatusRegisterOffset = $0000;  { to access specific functions }
AnalogWriteControlRegisterOffset = $1000;
AnalogWriteRegisterOffset = $3000;

BinaryControlRegisterOffset = $0000;
BinaryReadWriteRegisterOffset = $2000;
BinaryStatusRegisterOffset = $0000;

InterruptControlRegisterOffset :Word = $D000;
IssacDeviceNoRegisterOffset :Word = $C000;

AnalogDeviceCode = $0009;
BinaryDeviceCode = $0008;
DisableAllInterrupts = $0000;
EnableAnalogAD = $0001;
DisableAnalogAD = $0000;

NormalFlowCode = $64;
StorageFlowCode = $66;
NightFlowCode = $55;
CoolingSystemCode = $60;
DownDrainMainCode = $00;
DownDrainAuxillaryCode = $0A;

MaxUser = 10;
MaxData = 100;

Tab5 = ' ';

Var
CardNumber : 0..3;
InChar : char;
CheckInterruptStatus,
Channel,
ErrorCode,
Data,
X,
VoltageRange, VoltageOffset : integer;
Rdata, Volts : real;
Quit, TakeData, TextDirectMode, QuitProgram, ExitGraph, 
FirstTime, Update, InlineData : boolean;
Instr1, Instr2, 
NumberString : string[30];
BString, AString : IOType;

HousePred1, HousePred2, HousePred3, HousePred4, 
HouseConst, PanelPred1, PanelPred2, PanelPred3, PanelPred4, 
PanelConst, StoragePred1, StoragePred2, StoragePred3, 
StoragePred4, StorageConst, LightMax, LightDelta : Real;
DataIndex, Temperature : Integer;
SystemStartTime : Word;

ThermH, ThermP, ThermS, ThermL, 
FFTime1 : Array[1..100] of Integer;
SampleArray : SampleDataType;
HouseArray : PlotArray;

TextCharSet : TextCharArray;
X1TextRef, X2TextRef, Y1TextRef,
Y2TextRef : Integer;
BkColor : Byte;
User : Integer;
UserData : Array [0..MaxUser] of UserDataType;
UserDataPath : Str64;
UserString : String[3];
NumOfData : Integer;

Hour, Minute, Second, Sec100,
TC01, TRow,
CalTime,
CycleTime : Word;
PTime : Str11;
TPrint,
ShutDown : Boolean;
Dtemp : Word;
ATemp : Word;
Wtab : WtabArray;
HelpIndex : Word;

XDBar, RBar : Real;
XBarArray,
RangeArray : Array [1..20] of real;
n2, n3 : Integer;
n1, A2, D3, D4, d2 : TableData;

(==================================================================)

Implementation
Begin
CardNumber := 0;
X1TextRef := 0;
X2TextRef := 0;
Y1TextRef := 0;
Y2TextRef := 0;
Bkcolor := 1;
HousePredit1 := -73.7381;
HousePredit2 := -7.07285;
HousePredit3 := 10.16346;
HousePredit4 := -1.77828;
HouseConst := 227.2725;

PanelPredit1 := 78.43279;
PanelPredit2 := -62.3795;
PanelPredit3 := 15.08995;
PanelPredit4 := -1.45083;
PanelConst := 103.5134;

StoragePredit1 := 302.9336;
StoragePredit2 := -254.934;
StoragePred3 := 77.83091;
StoragePred4 := -8.26061;
StorageConst := 24.77193;

VoltageRange := 20;
VoltageOffset := 10;

LightMax := 4.4359 + 0.0051^0.043334;
LightDelta := 0.043334;

With UserData[0] do
  Begin
    LastName := 'Last Name';
    FirstName := 'First Name';
    Course := 'IEDT 231';
    TimesUsed := 1;
    Status := 'Student';
    Date := 'Last';
    Time := 'Last';
    Mode := 'Protected';
    Progress := 'Good / Bad';
    Password := 'Password';
    GradePoint := 3.99;
    Grade := 'A';
  End;

TCol := 5;
TRow := 25;
ExitGrap := False;
TPrint := True;
FirstTime := True;
InlineData := False;
TakeData := False;
ShutDown := False;
Update := False;
User := 0;
UserDataPath := 'A:\';
>UserString := '0';
DataIndex := 1;
Dtemp := 0;
BString := Binary;
AString := Analog;
HelpIndex := 0;
Randomize;
X := 1;
Repeat
  Dtemp := Random (MaxData*2);
  If (Dtemp > 50) and (Dtemp < MaxData*2) then
    Begin
      SampleArray[X] := Dtemp;
Inc(X);
End;
Until X > MaxData;
End. { unit }
Unit SPC_UTIL:

Interface

Uses
Crt.
Dos.
Turbo3.
SPC_ID.
MsUser.
MsScrnn.
SPC_t1T.
Introd.
GKernel.
GDriver;

Type
  Str8 = String[8];

Var
  DataFile : Text;
  RecordFile : File;
  IndexChannel : Integer;
  DataCounter : Integer;
  TextSize : Integer;

Function TestBit
(  var Target ;
   BitNum : integer
) : Boolean;
{ Test if bit in target is set (on) }
{ Target to test }
{ Bit number to test }
{ Function returns true if bit is set }

Function ConvertDataToVolts
(  RawData ; real;
   VoltSpan ,
   Offset : integer
) : real;
{ Convert raw counts to voltage }
{ Raw data to convert to voltage }
{ Total voltage range }
{ Negative offset of voltage range }
{ Function returns voltage value }

Function ConvertVoltsToData
(  Voltage : real;
   VoltSpan ,
   Offset : integer
) : integer;
{ Convert voltage to raw counts }
{ Voltage to convert to raw counts }
{ Total voltage range }
{ Negative offset of voltage range }
{ Function returns raw counts }

Function CardBaseAddress
(  CardNumber : integer
) : integer;
{ Convert CardNumber to base address }
{ Card number }
{ Function returns base address }
Procedure DisableBoardInterrupts
  ( CardNumber : integer )
  { Disable all interrupts. These modules work by polling and do not use interrupts. }
  { Card number to access }

Procedure InitDevice
  ( CardNumber : integer;
    Mode : IOtype )
  { Initialize specified data acquisition mode }
  { Card number to access }
  { Mode : ANALOG - for analog IO functions }
  { DIGITAL - for digital IO functions }

Function ReadAnalogData
  ( CardNumber, Channel : integer )
  : integer;
  { Routine for actual ANALOG read }
  { Channel number to access (0-3) }
  { Function returns value read }

Function ReadBinaryData
  ( CardNumber : integer )
  : integer;
  { Routine for actual DIGITAL read }
  { Card number to access }
  { Function returns the value read }

Procedure WriteAnalogData
  ( CardNumber, Channel, Value : integer );
  { Routine for actual ANALOG write }
  { Card number to access }
  { Channel number to access (0-1) }
  { Value to write }

Procedure WriteBinaryData
  ( CardNumber, Value : integer );
  { Routine for actual DIGITAL write }
  { Card number to access }

Function ConvertVoltsToTemperature:Integer;
  { Function converts volts read from analog channel to Temperature }
  { Function Returns a Long Integer Value for the Temperature. }

Procedure PrepareLogFile;

Procedure LogData;

Procedure LogTemperatureLightAndTime;

Procedure GetTemperature;

Procedure SwitchPump ( PumpNumber:Integer; PumpMode:String );
  { Switch either Pump 1 or Pump 2 on or off. PumpMode equals on/off }
  { PumpNumber select Pump 1 or 2. }

Procedure TurnSwitchOn ( SwitchNumber: Integer );
  { Switch any solenoid on to select direction of water flow }

Procedure TurnSwitchOff ( SwitchNumber: Integer );
( Switch any solenoid off to select direction of water flow )

Procedure ResetSystem; { Write 5 Volts to D/A port 0 for Interface Board to use for switching. Also Initialize the binary ports and set all ports to Logical 0 [0 volts]. }

Procedure AdjustSystem;

Procedure ShutDownSystem; (This routine will shut the system down by sending 0 to the binary ports and, 0 Volts to the D/A port used for switching.)

(*******************************************************************)

Implementation

Function TestBit
( var Target :
  BitNum : integer
) : Boolean; { Function returns true if bit is set }

var
  Subject : Integer absolute Target;
  Temp : integer;

begin
  Temp := Subject;
  if BitNum > 0 then Temp := Temp SHR BitNum;
  if Odd(Temp) then TestBit := True
  else TestBit := False;
end; { TestBit }

Function ConvertDataToVolts
( RawData : real;
  VoltSpan, Offset : integer
) : real;

begin
  ConvertDataToVolts := (RawData * VoltSpan / 4096) - Offset;
end;

Function ConvertVoltsToData
( Voltage : real;
  VoltSpan, Offset : integer
) : integer;

begin
ConvertVoltsToData := Round ((4096 * (Rdata + Offset)) / VoltSpan);
end;

Function CardBaseAddress (CardNumber : integer) : integer;

begin
Case CardNumber of
0 : CardBaseAddress := $02E2;
1 : CardBaseAddress := $06E2;
2 : CardBaseAddress := $0AE2;
3 : CardBaseAddress := $0EE2;
end; { CardBaseAddress }
end: { CardBaseAddress }

Procedure DisableBoardInterrupts (CardNumber : integer);

var
BaseAddress,
CheckInterruptStatus : integer;

begin
BaseAddress := CardBaseAddress(CardNumber);
Port[BaseAddress+InterruptControlRegisterOffset] := DisableAllInterrupts;
Delay(1);
CheckInterruptStatus := Port[BaseAddress+InterruptControlRegisterOffset];
ClrScr;
if (CheckInterruptStatus AND 7) <> 0 then begin
begin
WriteLn('Interrupts NOT cleared.');
WriteLn('CheckInterruptStatus = ',CheckInterruptStatus);
WriteLn('Exiting Program.');
Halt;
end;
end; { DisableBoardInterrupts }

Procedure InitDevice (CardNumber : integer; Mode : I0type);

var
BaseAddress : integer;

begin
...
begin
BaseAddress := CardBaseAddress(CardNumber);
end Mode of
  Analog : Port[BaseAddress+IssacDeviceNoRegisterOffset] := AnalogDeviceCode;
  Binary : Port[BaseAddress+IssacDeviceNoRegisterOffset] := BinaryDeviceCode;
end; { case }
Delay(1);
end; { InitDevice }

Function ReadAnalogData { Routine for actual ANALOG read }
  ( CardNumber, ( Card number to access )
    Channel : integer ( Channel number to access (0-3) )
    ) : integer ( Function returns value read )
var
  BaseAddress,
  LowByteData,
  HighByteData,
  TestStatus : integer;
begin
BaseAddress := CardBaseAddress(CardNumber):
Port[BaseAddress+AnalogReadControlRegisterOffset] := DisableAnalogAD;
Port[ BaseAddress
  + AnalogReadControlRegisterOffset
  + SelectHighOrderByte ] := Channel; { set A/D channel }
Delay(1); { *** Wait for channel multiplexer to settle *** }
  { Delay(1) is equivalent to approximately 1000 microseconds on 4.77 MHz CPU. An actual delay of only 120 microseconds is required here for the first time the channel is selected }
Port[ BaseAddress
  + AnalogReadControlRegisterOffset ] := EnableAnalogAD; { enable A/D }
  { conversion }
Port[ BaseAddress
  + AnalogReadControlRegisterOffset
  + SelectHighOrderByte ] := Channel; { set A/D channel }
Delay(1); { *** Wait for channel multiplexer to settle *** }
  { An actual delay of only 20 microseconds is required here since the channel has already been selected }
{ Test for completion of A/D conversion }

repeat
  TestStatus := Port[BaseAddress + AnalogStatusRegisterOffset];
  TestStatus := TestStatus AND 1; { mask out all but A/D busy bit }
until TestStatus = 0;

{ Force A/D converter to read status }
Port[BaseAddress + AnalogReadControlRegisterOffset] := DisableAnalog;

{ Set channel to read }
Port[BaseAddress + AnalogReadControlRegisterOffset + SelectHighOrderByte] := Channel;

{ Read A/D data }
LowByteData := Port[BaseAddress + AnalogReadRegisterOffset]; { read }
HiByteData := Port[BaseAddress + AnalogReadRegisterOffset + SelectHighOrderByte]; { data }

ReadAnalogData := (HiByteData SHR 8) + LowByteData;
end; { ReadAnalogData }

Function ReadBinaryData(CardNumber: integer) { Routine for actual DIGITAL read } integer;
  { Card number to access } Function returns the value read
begin
  BaseAddress := CardBaseAddress(CardNumber);
  LoByteValue := Port[BaseAddress + BinaryReadWriteRegisterOffset];
  HiByteValue := Port[BaseAddress + BinaryReadWriteRegisterOffset + SelectHighOrderByte];
  ReadBinaryData := (HiByteValue SHR 8) + LoByteValue;
end; { ReadBinaryData }

Procedure WriteAnalogData(CardNumber: integer; Channel: integer; Value: integer); { Routine for actual ANALOG write } { Card number to access } { Channel number to access (0-1) } { Value to write }

var
BaseAddress : integer;

begin
BaseAddress := CardBaseAddress(CardNumber);

Port[B BaseAddress + AnalogWriteControlRegisterOffset + SelectHighOrderByte] := Channel;

Port[B BaseAddress + AnalogWriteRegisterOffset] := Lo(Value);

Port[B BaseAddress + AnalogWriteRegisterOffset + SelectHighOrderByte] := Hi(Value);
end; { WriteAnalogData }

Procedure WriteBinaryData (CardNumber, Value : integer);
{ Routine for actual DIGITAL write }
{ Card number to access }

var
BaseAddress : integer;

begin
BaseAddress := CardBaseAddress(CardNumber);

Port[B BaseAddress + BinaryReadWriteRegisterOffset] := Lo(Value);

Port[B BaseAddress + BinaryReadWriteRegisterOffset + SelectHighOrderByte] := Hi(Value);
end; { WriteBinaryData }

Function ConvertVoltsToTemperature:Integer; { Function converts volts read from analog channel to Temperature Function Returns a Long Integer Value for the Temperature. }

Var Y : Real;

Begin
{ Rdata := ReadAnalogData(CardNumber, Channel);
  Rdata := ConvertDataToVolts(Rdata, VoltageRange,}
VoltageOffset );

Case IndexChannel of
  0 : Y := ((LightMax - Rdata) / LightDelta) ; {Calculate Light Intensity }
        { Calculate Panel Temperature }
  1 : Y := PanelPred1* Rdata + PanelPred2*Sqr(Rdata) + PanelPred3*Rdata*Sqr(Rdata)
      + PanelPred4*Sqr(Sqr(Rdata)) + PanelConst ;
  2 : Y := HousePred1* Rdata + HousePred2*Sqr(Rdata) + HousePred3*Rdata*Sqr(Rdata)
      + HousePred4*Sqr(Sqr(Rdata)) + HouseConst ;
        { Calculate House Temperature }
  3 : Y := StoragePred1* Rdata + StoragePred2*Sqr(Rdata) + StoragePred3*Rdata*Sqr(Rdata)
      + StoragePred4*Sqr(Sqr(Rdata)) + StorageConst ;
        { Calculate Storage Temperature }
End;

If IndexChannel = 0 then
  ConvertVoltsToTemperature := Trunc(Y)
Else
  ConvertVoltsToTemperature := Round(Y);
End;

Procedure PrepareLogFile;
Var
  S: String[2];

Begin
  Str(User, S);
  Assign(DataFile, UserDataPath+ 'USER'+ S + '.&
  If FirstTime Then Rewrite(DataFile)
  Else
    ($I- )
    Append(DataFile);
    ($I+ )
  If IOResult <> 0 then
    Begin
      SelectWindow(9);
      SetBackground(0);
      SelectSpc;
      SelectWindow(10);
      GotoXY(2, 24);
      Write(#7#7,'File [' UserDataPath+ 'USER'+ S + '.&
      ' does not exist',
      ' will create a new file by this name');
      Rewrite(DataFile);
      Delay(800);
      SelectWindow(9);
      SetBackground(0);
      SetHeaderOff;
DrawBorder;
SetHeaderOn;
End;

WriteLn(DataFile, 'CenterText(80,'Temperature and Light Intensity Read + From System'));
WriteLn(DataFile, ' ');
PrintTm(Tcol,TRow,PTime,False);
WriteLn(DataFile,' Starting time = ',PTime);
WriteLn(DataFile, ' ');
WriteLn(DataFile,' Time',Tab5,' House',Tab5,' Panel',Tab5,' Storage',Tab5,' Light Intensity');
WriteLn(DataFile, ' ');
DataIndex := 1;
FirstTime := False;
End;

Procedure LogData;
var S:Integer;
Begin
WriteLn(DataFile, ' ');
S := DataIndex;
WriteLn(DataFile, ':1,CalTime,' ':2,ThermH[s]:2,' ':1, Tab5:5,' ':2,ThermP[s]:2,' ':1, Tab5:5,' ':3,ThermS[s]:2,
 ':5,Tab5:5,Tab5:5,ThermL[s]);
End:

Procedure LogTemperatureLightAndTime;
Var
 I : Integer;
 ch : Char;
Begin
 Quit := False;
 TakeData := False;
 I := 1;
 SelectSPC;
 Inc(Dtemp);
 TakeData := (CycleTime in [10,20,30,40,50,60,70,80,90]);
 GotoXY(25,25);
 SelectWindow(1);
 Write ('Time Elapsed:',CalTime,'s':1);
 If CycleTime = 0 then
 Begin
   GotoXY(25,25);
   Write (' ');
 End;
 GotoXY(52,24);
 Write ('Cycle Time:':11,CycleTime,'s':1);
 TCol := 2;
 TRow := 2;
PrintTime(TCol, TRow, PTime, TPrint);
If (TakeData) Then
  Begin
    Case IndexChannel of
      0: ThermL[DataIndex] := Temperature;
      1: ThermP[DataIndex] := Temperature;
      2: ThermH[DataIndex] := Temperature;
      3: ThermS[DataIndex] := Temperature;
    End;
    FFTime1[DataIndex] := CalTime;
    DTemp := CalTime;
    DataCounter := CycleTime;
    Inc(DataIndex);
    WaitTime(1);
  End;
  If DataIndex >= 100 then
    Begin
      {$-1} 
      Close(DataFile);
      ExitGraph := True;
      Quit := True;
      {$+1} 
    End;
  end; { LogTemperatureLightAndTime }

Procedure GetTemperature;

var Index, Y : Integer;
Symbol : String[3];
TempString : String;

begin
  SelectWorld(4);
  SelectWindow(7);
  If ShutDown Then { Reopen file if system is back to normal 
    and set shutdown to false for a recheck of system }
    Begin
      Append(DataFile);
      Shutdown := False;
    End;
  SelectSPC;
  GotoXY (55, 15); Write ('Light Intensity ', #16);
  GotoXY (55, 16); Write ('Temp. of Panel ', #16);
  GotoXY (55, 17); Write ('Temp. of House ', #16);
  GotoXY (55, 18); Write ('Temp. of storage', #16);
  InitDevice(CardNumber, AString);
  For Index := 0 to 3 do
    Begin

IndexChannel := Index;
Rdata := ReadAnalogData(CardNumber, Index);
GetTime(Hour, Minute, Second, Sec100);
CalTime := Hour*Sqr(60)+Minute*60+Second;
Rdata := ConvertDataToVolts(Rdata, VoltageRange, VoltageOffset);
Temperature := ConvertVoltsToTemperature;

GotoXY(73, 15+Index);
If Index = 0 then
  Symbol := ' %'
Else
  Symbol := ' xF';
If (Temperature <= 50) (Or (Rdata >= 2.0)) then
  ShutDown := True;
write(Temperature:3, Symbol:3);
LogTemperatureLightAndTime;
End;
If (TakeData) And (Not ShutDown) then
Begin
  LogData;
  TakeData := False;
End;
end; { GetTemperature }

Procedure SwitchPump (PumpNumber:Integer; PumpMode:String);
{ Switch either Pump 1 or Pump 2 on or off. PumpMode equals on/off
  PumpNumber select Pump 1 or 2. }
var Switch, SwitchData: Integer;
begin
  SwitchData := ReadBinaryData(CardNumber);
  IF PumpNumber = 1 then
    IF PumpMode = 'ON' then
      WriteBinaryData(CardNumber, SwitchData or 16)
    ELSE
      WriteBinaryData(CardNumber, SwitchData and 111);
  IF PumpNumber = 2 Then
    IF PumpMode = 'ON' then
      WriteBinaryData(CardNumber, SwitchData or 32)
    ELSE
      WriteBinaryData(CardNumber, SwitchData and 95);
end; { SwitchPump }

Procedure TurnSwitchOn (SwitchNumber: Integer );
{ Switch any solenoid on to select direction of water flow }
Var SwitchData : Integer;
Begin

SwitchData := ReadBinaryData(CardNumber);

Case SwitchNumber of
  1 : WriteBinaryData (CardNumber, SwitchData OR 1);
  2 : WriteBinaryData (CardNumber, SwitchData OR 2);
  3 : WriteBinaryData (CardNumber, SwitchData OR 4);
  4 : WriteBinaryData (CardNumber, SwitchData OR 8);
  5 : WriteBinaryData (CardNumber, SwitchData OR 64);
  6 : WriteBinaryData (CardNumber, SwitchData OR 256); { Turn BO 8 off }
End: { Case }
End: { TurnSwitchOn }

Procedure TurnSwitchOff (SwitchNumber: Integer):
{ Switch any solenoid off to select direction of water flow }

Var SwitchData : Integer;

Begin

SwitchData := ReadBinaryData(CardNumber):

Case SwitchNumber of
  1 : WriteBinaryData (CardNumber, SwitchData AND 126);
  2 : WriteBinaryData (CardNumber, SwitchData AND 125);
  3 : WriteBinaryData (CardNumber, SwitchData AND 123);
  4 : WriteBinaryData (CardNumber, SwitchData AND 119);
  5 : WriteBinaryData (CardNumber, SwitchData AND 128);
  6 : WriteBinaryData (CardNumber, SwitchData AND 255); { Turn BO 8 off }
End: { Case }
End: { TurnSwitchOff }

Procedure ResetSystem; { Write 5 Volts to D/A port O for Interface Board to use for switching. Also Initialize the binary ports and set all ports to Logical 0 [0 volts]. }

begin
  Channel := 0;
  InitDevice(CardNumber, AString);
  Rdata := 5;
  Data := ConvertVoltsToData(Rdata, VoltageRange, VoltageOffset);
  WriteAnalogData(CardNumber, Channel, Data);
  InitDevice(CardNumber, BString);
  WriteBinaryData (CardNumber, 0);
  TurnSwitchOn(6);
  TurnSwitchOn(5);
end: { ResetSystem }
Procedure AdjustSystem;

Begin { AdjustSystem }
  If ShutDown Then
    Begin
      WriteBinaryData (CardNumber, 0);
      ResetSystem;
    End;
  End: { AdjustSystem }

Procedure ShutDownSystem: (This routine will shut the system down
  by sending 0 to the binary ports and,
  0 Volts to the D/A port used for switching.)

Var  I :Integer;

Begin
  InitDevice(CardNumber,BString);
  SwitchPump (1,'Off');
  WaitTime(3);
  SwitchPump (2,'Off');
  WaitTime(3);
  For I := 1 to 4 do
    begin
      TurnSwitchOff(I);
      WaitTime(3);
    end;
  TurnSwitchOff(6);
  WaitTime(3);
  TurnSwitchOff(5);
  WaitTime(3);
  WriteBinaryData (CardNumber, 0);
  InitDevice(CardNumber, AString);
  Rdata := 0;
  Data := ConvertVoltsToData(Rdata, VoltageRange, VoltageOffset);
  WriteAnalogData(CardNumber, Channel, Data);
End;
Begin
  TextSize := 1;
End. { unit }
Unit SPC_Tit;

Interface

Uses Dos,Crt,Turbo3,Introd,GDriver,GKernel,GWindow{,GShell};

Var Choice :Integer;

procedure delay(n: real);

procedure ClearEol(i:integer);

procedure SelectSPC:

procedure DefineWindowSPC(i,X1,Y1,X2,Y2:integer);

procedure msg(s:wrkstring);

procedure waitreturn (N:Word);

Procedure TitlePage;

Procedure Title;

procedure delay(n: real);

{====================================================================}

procedure delay(n: real);
var i:real;
  j:integer;
  ch:char;
  quit:boolean;

begin
  i:=0;
  ch:='';
  repeat
    i:=i+1;
    quit:=false;
    if keypressed then
      begin
        Choice := 10;
        ch := readKey;
        quit:=(ch='+C);
        if (ch='$') and keypressed then
          begin
            ch := readKey;
            quit:=(ch='0');
            ch:=' ';
if quit then
begin
Choice := 10;
leavegraphic;
halt;
end;
until (ch='M') or (i>=n);
end;

procedure ClearEol(i:integer);
begin
  gotoxy(2,i);
  write(' ');
end;

procedure DefineWindowSPC(i1,X1,Y1,X2,Y2:integer);
begin
  DefineWindow(i,Trunc(X1/79*XMaxGb+0.001),Trunc(Y1/199*YMaxGb+0.001),
      Trunc(X2/79*XMaxGb+0.5),Trunc(Y2/199*YMaxGb+0.5));
end;

procedure msg(s:wrkstring);
begin
  DefineWorld(3.0,MaxWorldY,MaxWorldX,0); { Define this World to Draw in }
  SelectWorld(3);
  defineWindowSPC(9,1,YMaxGb-21,XMaxGb-1,YMaxGb-1);
  SelectWindow(9);
  DrawBorder;
  gotoxy(30,24);
  write(s);
  delay(750);
end;

procedure waitreturn( N:Word);
begin
  Msg('Press RETURN to continue ...');
  Delay (N);
end;

procedure SelectSPC;
const
  MaxXWorld:Float = 110.0;
MaxYWorld: Float = 180.0;
begin
  DefineWindowSPC (10.0,0.XMaxGlb, YMaxGlb);
  DefineWindow(1.0.0,MaxXWorld,MaxYWorld);
  SelectWorld(1);
  SelectWindow(10);
end;

{-----------------------------------------------}

Procedure TitlePage;

Begin (TitlePage)
  SetHeaderOff;
  DefineWorld(2.0.0,639.199);
  DefineWindow(1.5,5.XMaxGlb-5, YMaxGlb-25);
  SelectWorld(2);
  SelectWindow(1);
  SetForegroundColor(Yellow);
  SetBackgroundColor(Blue);
  DrawBorder;
  TextColor(Yellow);
  TextBackground(Blue);
  DrawText(1,30.8.CenterText(12.'ROPE'));
  DrawText(1,80.3.CenterText(36.'A SPC Instructional Package'));
  DrawText(1,100.2.CenterText(50.'By Renford A. B. Brevett'));
  DrawText(1,140,3.CenterText(36.'Iowa State University (c)'));
  DrawText(1,160,2.CenterText(48.'June 1988'));
  SetHeaderOn;
End; (TitlePage)

{-----------------------------------------------}

Procedure Title;

Begin
  InitGraphic;
  DefineWorld(2.20,10.180,630);
  DefineWindow(10,20.10.XMaxGlb-20, YMaxGlb-10);
  TitlePage;
  Choice := 0;
  Repeat
    WaitReturn(25000);
    SwapScreen;
    WaitReturn(25000);
    SwapScreen;
    Until Choice = 10;
  LeaveGraphic;
End;

end. (Unit)
Unit INTROD;

Interface

Uses Dos,Crt,Turbo3,MsScrnn1,MsUser,Spc_ID;

const
Up     = true;
Down   = false;

IntDoc: Array [0..35] of Str80 =
('','
Welcome to ROPE (Real-time On-line Processing Environment) at ';
'Iowa State University Ames Iowa. This package was designed by ';
'Renford Brevett as part of his masters thesis. It was conducted',
'under the guidance of his major professor William G. Miller and ';
'also Professor John Even Jr. of the Industrial Engineering ';
'Department and Professor William Wolansky of the Industrial ';
'Education and Technology Department.';'
'$',
'
Questions or comments can be addressed to:','
'
W. G. Miller',
'Industrial Education And Technology',
'Iowa State University',
'Ames, Iowa 50011.',
'Ph. (515)-294-6964'.
'$',
'This package is designed as a tool for a real-time on-line ',
'computer-based demonstration of Statistical Process Control (SPC).','
'This package is designed to be used by, but not limited to ',
'undergraduate Industrial Technology, Industrial Education ',
'and Industrial Engineering students at Iowa State University. ',
'It is easy to understand the concepts of SPC that are discussed ',
'and illustrated in this package, since the knowledge of only ','introductory statistics is assumed.','
'$',
'
This program is menu driven with consistency in the window ',
'display. Since computer-based instruction is unique, and',
'computerization has proliferated in the technical world, the(',
'opportunity to gain close aquaintance with this area will be',
'advantageous. ',
'$',
'"');
type MaxString = string [255];

var LineNum,I,Win,StringLen: Integer;
Ch : char;
Quit,NextScreen : Boolean;
WP :WindowRec;

Procedure DefineTextWindow(TWin,_TX1,TY1,TX2,TY2:Integer{.TextTitle:String[80]} );
{ This Unit defines a window. }

procedure SelWindow(Win: Integer);
Procedure ShowTextWindow(SHWin:Integer);
Procedure CleanWindow(Win:Integer);
Procedure Beep;
Procedure Beep2;
Procedure ClearWindow (Xposn,Yposn,XSize,Ysize,Attr:Byte);

Function FetchTime:Word;
{ Used to get the time and return the current time in seconds }

Procedure WaitTime(Limit:Integer);
{ Delay for Limit seconds }

Procedure PrintTime(TCol,TRow:Integer;
Var PTime:Str11;TPrint:Boolean);
{ Print the time at location TRow,TCol if TPrint is true. Also return the string PTime with the Time in the format (hh:mm:ss) }

Function CenterText (FieldWidth:Integer;CenterString:Str80):Str80;

procedure UpdateWindow;

procedure Introduction;

Procedure ProcessIntro:
{********************************************}

Implementation
Procedure DefineTextWindow(TW, TX1, TY1, TX2, TY2: Integer);
{ This Unit defines a window. }
Begin
  Wtab[TW, 1] := TX1;
  Wtab[TW, 2] := TY1;
  Wtab[TW, 3] := TX2;
  Wtab[TW, 4] := TY2;
  Wtab[TW, 5] := 1;
End;

procedure SelWindow(Win: Integer);

begin
  Window(Wtab[Win, 1]+1, Wtab[Win, 2]+1, (Wtab[Win, 3]+WTab[win, 1])-2,
        (Wtab[Win, 4]+WTab[Win, 2])-2);
  GotoXY(1, 1);
end; { SelWindow }

Procedure ShowTextWindow(SHWin: Integer);
Var
  Xsize, Ysize : Integer;

Begin
  Window(1, 1, 80, 25);
  Xsize := Wtab[SHWin, 3] + 1;
  Ysize := Wtab[SHWin, 4] + 1;
  EdSetCursor(CursorOff);
  EdDrawBox(Border, Wtab[SHWin, 1]-1, Wtab[SHWin, 2]-1,
            Xsize, Ysize, NormalBox);
  SelWindow(SHWin);
  GotoXY(1, 1);
End;

Procedure CleanWindow(Win: Integer);
Var CCol, CRows : Integer;

Begin
  TextBackGround(BkColor);
  ClearWindow(Wtab[Win, 1]-1, Wtab[Win, 2]-1, Wtab[Win, 3]+2,
              Wtab[Win, 4]+2, BkColor*16);
  SelWindow(Win);
End;

Procedure Beep;

begin
Sound(500);
Delay(300);
NoSound;
end; { Procedure Beep }

Procedure Beep2;
begin
  Sound(1500);
  Delay(200);
  Sound(3300);
  Delay(400);
  NoSound;
end; { Procedure Beep2 }

Procedure ClearWindow (Xposn,Yposn,XSize,Ysize,Attr:Byte):
Var S: String;
  I: Integer;
Begin
  S[0] := Chr(XSize-4);
  FillChar (s[1].Length(s), chr($20));
  For I := 1 to YSize - 3 Do
    EdFastWrite ( s. YPosn + I, Succ(XPosn), Attr);
End;

Function FetchTime:Word;
  { Used to get the time and return the current time in seconds }
Begin
  GetTime(Hour,Minute,Second,Sec100);
  FetchTime := Hour*Sqr(60)+Minute*60+Second;
End:

Procedure WaitTime(Limit:Integer);
  { Delay for Limit seconds }
Var StartTime,EndTime:Word;
Begin { WaitTime }
  StartTime := FetchTime;
  Repeat
    EndTime := FetchTime;
    Until (EndTime-StartTime) >= Limit;
End { WaitTime }

Procedure PrintTime(TCol,TRow:Integer;Var
PTime:Str11:TPrint:Boolean):
    { Print the time at location TRow,TCol if TPrint
     is true. Also return the string PTime with the
     Time in the format (hh:mm:ss) }

var PHour :Word;
    Mode :String[3];
    H,M,S :String[2];

Begin

    GetTime(Hour,Minute,Second,Sec100);
    If Hour = 0 Then
        Begin
            PHour := 12;
            Mode := 'am';
        End
    Else
        If Hour >= 13 Then
            Begin
                PHour := Hour - 12;
                Mode := 'pm';
            End
        Else
            Begin
                PHour := Hour;
                Mode := 'am';
            End;
    End;

    If PHour > 12 then Mode := 'pm';
    If PHour = 0 then Mode := 'am';
    If Hour = 0 then Mode := 'am';
    If (Minute = 0) and (Second < 6 ) then Beep2;
    If (Minute in [15,45]) and (Second < 6 ) Then beep;
    If (Minute = 30) and (Second < 6 ) then
        Begin
            Beep;
            Beep2;
        End;

    Str(PHour,H);Str(Minute,M);Str(Second,S);
    If Minute < 10 Then M := '0'+M;
    If Second < 10 Then S := '0'+S;
    PTime := H+':'+M+':'+S+Mode;

    If TPrint Then
        Begin
            GotoXY(TCol,TRow);
            Write(H:2,':'+M:2,':'+S:2,Mode:2);
        End;
End;
Function CenterText (FieldWidth:Integer;CenterString:StrSO):StrSO;
var
  TextSpacer:StrSO;
  J,JLen :Integer:
Begin
  JLen := Length(CenterString);
  If JLen < FieldWidth then
    Begin
      FillChar(TextSpacer,FieldWidth +1,$20);
      TextSpacer[0] := Chr(FieldWidth);
      J := (FieldWidth - JLen) Div 2;
      Move (CenterString[1],TextSpacer[J+1],JLen);
      CenterText := TextSpacer;
    End
  Else
    CenterText := Copy(CenterString,1,FieldWidth);
  End; { CenterText }

procedure UpdateWindow:
VAR Y:INTEGER;
begin
  NormVideo;Win := 3;BkClr := 1;
  if intdoc[LineNum] = '#' then
    Quit := True
  else
    if intdoc[LineNum] = '$' then
      begin
        TextColor(Yellow);
        SelWindow(1);
        gotoXY(15,2);
        Edfastwrite ('---> Press any key to continue <<< ' ,25,15,159);
        repeat
          ShowTextWindow(Win);
        until keystpressed;
        Edfastwrite (' ',25,15,0);
        NextScreen := True;
        ch := Readkey;
        CleanWindow(3);
      end
    else
      Begin
        SelWindow(Win);
        EdFastWrite(INTDOC[LineNum],WTAB[Win,2] + I,WTAB[Win,1]+4,
              Yellow+Blue*16);
        If (LineNum > 11) and (LineNum < 18) then
          I := I + 1
      end
end;
else
I := I + 2;
{ ShowTextWindow(Win);} 
end;
NormVideo;
end; {UpdateWindow}

procedure Introduction;
var HeapPtr : Integer;
begin
Mark(HeapPtr);
Wtab[3, 1] := 3;
Wtab[3, 2] := 3;
Wtab[3, 3] := 74;
Wtab[3, 4] := 20;
Wtab[3, 5] := 1;
Win := 3;
LineNum := 0;
NextScreen := False;
I := 0;
TextColor(yellow);
TextBackground(blue);
BkColor := 1;
CleanWindow(3);
SelWindow(3);
Quit := False;
EdSetCursor(CursorOff);
EdDrawBox(Border, Wtab[Win, 1] - 1, Wtab[Win, 2] - 1, Wtab[Win, 3] + 1,
       Wtab[Win, 4] + 1, NormalBox);
repeat
   If NextScreen then
   begin
      I := 0;
      NextScreen := False;
   end;
   UpdateWindow;
   LineNum := LineNum + 1;
until Quit:
Window(1, 1, 80, 25);
ClrScr;
GotoXY(1, 24);
Release(HeapPtr);
end; {Introduction}

Procedure ProcessIntroduction;

Begin
TextColor(Yellow);
TextBackground(Red);
EdSaveTextWindow(Border, 'What is RDPE', 5, 5, 75, 20, WP);
DefineTextWindow(6,6,69,14);
CleanWindow(6);
WriteLn(' ROPE is an interactive environment where you can observe');
WriteLn('graphically the changes within an industrial process as');
WriteLn('changes are made to its operating variables. This process');
WriteLn('is a Solar Collector Heat Exchange Unit (SCHEU). The ‘');
WriteLn('variables that can be changed by the user are the Limits of’);
WriteLn('control (2e, 3e, 4e etc..), sampling time (in seconds), and’);
WriteLn('the cycle time of the process (in seconds). ‘');
WriteLn;
Ch := ReadKey;
EdRestoreTextWindow(WP);
End;

Procedure ProcessInstruction;
Var
    Ch:Char;
Begin
    TextColor(Yellow);
    TextBackground(Red);
    EdSaveTextWindow(Border,'How To Use ROPE',5,5,75,20,WP);
    DefineTextWindow(6,6,69,14);
    CleanWindow(6);
    WriteLn(' To use the ROPE environment the system must be ready ‘);
    WriteLn('to run. Be sure all plugs to system are plugged into the wall.’);
    WriteLn('Also set computer override switch to COMPUTER. Switch power’);
    WriteLn('supply ON. At this point nothing should happened but if’);
    WriteLn('any device is ON the computer will first turn them off.’);
    WriteLn('The bottom of the screen shows instructions on how to change’);
    WriteLn('variables and other settings.’);
    WriteLn;
    WriteLn(' << Press Any Key to Continue >> ‘);
    Ch := ReadKey;
    EdRestoreTextWindow(WP);
End;

Procedure ProcessIntro;
var
    ProcessPtr :#Integer;
Begin
    { Mark(ProcessPtr); }
    ProcessIntroduction;
    ProcessInstruction;
    { Release(ProcessPtr); }
End;
Begin
  Clrscr;
  GotoXY(25,12);
  TextBackground(Blue);
  TextColor(Yellow+Blink);
  Write('Loading Program Please wait ...');
End.
Unit SPC_Rec:

Interface

Uses
  Crt,
  Dos,
  SPC_Id,
  MsUser,
  MsScrnn1,
  MsString,
  Introd,
  SPC_Help;

var
  Count,Result :Word;
  RecordFile,RecordFile2 :File Of UserDataType;
  NewUser :UserDataType;
  UserList :Array[1..MaxUser] of UserDataType;
  Ch :char;
  Quit,'?
  EndData,
  Profile :Boolean;
  Changes :String[3];
  UserName :String;
  UserPassword :String;
  SFPassword,SFLastName :String;
  M :WindowRec;
  WrongPasswordTry :Integer;
  MajorUser :Integer;
  PTime :Str11;

Procedure CodeVar(Var S:Str14;PWField:Integer);

Procedure DeCodeVar(Var S:Str14;PWField:Integer);

Procedure SetRecords;

Procedure GetRecordinfo;

Procedure NameRecords;

Procedure FindRecords;

Procedure SearchForInfo;

Procedure UpdateInfo;

Procedure MakeScreen;
Procedure GetNameAndPword;
Procedure ListUsers;
Procedure DeleteUser;
Procedure ChangeInfo;
Procedure AddUser;
Procedure GetUserStatus;

(*=================================================================*)

Implementation

Procedure CodeVar(Var S:Str14; PWField:Integer);
{ Code is protected by Renford A. B. Brevett }
Procedure DeCodeVar(Var S:Str14; PWField:Integer);
{ Code is protected by Renford A. B. Brevett }
Procedure UpdateInfo;
var S:Str14;
Begin
ShowTextWindow(2);
BkColor := Red;
ClearWindow(2);
TextColor( Yellow);
SelWindow(2);
With UserData[ User ] Do
Begin
FillChar(S, Length(Password), '*');
S[0] := Password[0];
GotoXY(2,2); Write ('Name : ', FirstName, ', LastName);
GotoXY(2,3); Write ('Course : ', '13.Course);
GotoXY(2,4); Write ('Times Used : ', '13.TimesUsed);
GotoXY(2,5); Write ('Status : ', '13.Status);
GotoXY(2,6); Write ('Mode : ', '13.Mode);
GotoXY(2,7); Write (' Last Used : ', '13);
GotoXY(2,8); Write (' MMMMMMMM ', '13);
GotoXY(2,9);Write (' Date : ', '13.Date);
GotoXY(2,10);Write ('Time : ', '13.Time);
GotoXY(2,11);Write ('Progress : ', '13.Progress);
GotoXY(2,12);Write ('Password : ', '13.S.', '*');
GotoXY(2,13);Write ('G.P.A. : ', '13.GradePoint:5:2');
Write (Tab5,' Grade <', Grade, '>');
End;
Procedure ExitRecAndUpdate;
Var Year,Month,Day,DOW :Word;
Y,M,D,DW :String;
Begin
    GetDate(Year,Month,Day,DOW);
    Str(Year,Y);
    Str(Month,M);
    Str(Day,D);
    PrintTime(TCol,TRow,PTime,False);
    UserData[MajorUser].TimesUsed := UserData[MajorUser].TimesUsed + 1;
    UserData[MajorUser].Date := M+'/'+D+'/'+Y;
    UserData[MajorUser].Time := PTime;
    {$!-
    Assign(RecordFile,'ROPE.LST');
    Reset (RecordFile);
    Seek(RecordFile,User);
    Write (RecordFile,UserData[MajorUser]);
    {$!+
    If IOResult <> 0 then
        Write ('Error Opening File ROPE.LST');
    Close (RecordFile);
End;

Procedure SetRecords;
Begin
    With NewUser Do
        Begin
            Date := '05/13/88';
            Time := '01:33:am';
            Progress := 'First Time User';
            Grade := '?';
            GradePoint := 0.00;
            TimesUsed := 0;
        End;
End;

Procedure GetRecordInfo;

Var
    Ch :Char;
Begin
    With NewUser do
        Begin
            GotoXY(5,24);ClrEol;Write('Enter Last Name..');
            Ch := ReadKey;
            If Ch = ESC then EndData := True
        End;
    End;
Else
Begin
    EndData := False;
    write(Ch);
    Readln(LastName);
    LastName := Ch + LastName;
    GotoXY(5.24);ClrEol;
    Write('Enter First Name..') : Readln(FIrstName);
    GotoXY(5.24);ClrEol;
    Write('Enter SS Number..') : Readln(SSNumber);
    GotoXY(5.24);ClrEol;
    Write('Enter Course..') : Readln(Course);
    GotoXY(5.24);ClrEol;
    Write('Enter Status..') : Readln(Status);
    If Status = 'Professor' then Profile := True
    Else Profile := False;
    GotoXY(5.24);ClrEol;
    Write('Enter Mode..') : Readln(Mode);
    GotoXY(5.24);ClrEol;
    Write('Enter Password..') : Readln(Password);
    CodeVar(Password, Length(Password));
    SetRecords;
    Writeln;
End;
End;

Procedure NameRecords;
var I : Integer;
Begin
    Assign (RecordFile2, 'ROPEPF.RES');
    ($I-)
    Erase(RecordFile2);
    Assign (RecordFile2, 'ROPEPF.LST');
    Rename (RecordFile2, 'ROPEPF.RES');
    Assign (RecordFile2, 'ROPEPF.LST');
    Rewrite (RecordFile2);
    Assign (RecordFile, 'ROPE.RES');
    ($I-)
    Erase(RecordFile);
    ($I+)
    Assign (RecordFile, 'ROPE.LST');
    Rename (RecordFile, 'ROPE.RES');
    Assign (RecordFile, 'ROPE.LST');
    Rewrite (RecordFile);
    ShowTextWindow(2);
    CleanWindow(2);
    SelWindow(2);
    Writeln(' Enter Data For Each User ');
WriteIn(' Press ESC when Done ');
I := 0;
EndData := False;
CleanWindow(3);
SelWindow(3);
While Not EndData do
Begin
  GetRecordInfo;
  If Profile Then
    Write (RecordFile2, NewUser)
  Else Write (RecordFile, NewUser);
  Inc(I);
End;
Close (RecordFile2);
Close (RecordFile);
End;

Procedure FindRecords;
Var I:Integer;
  Ch:Char;
Begin
  Assign (RecordFile, 'ROPE.LST');
  Reset (RecordFile);
  SelWindow(2);
  I := 0;
  While Not Eof(RecordFile) do
  Begin
    Seek(RecordFile, I);
    Read(RecordFile, UserData[I]);
    With UserData[I] do
    Begin
      GotoXY(5, 2); ClrEol; Write(' Last Name.. ', LastName);
      GotoXY(5, 3); ClrEol; Write(' First Name.. ', FirstName);
      GotoXY(5, 4); ClrEol; Write(' SS Number.. ', SSNumber);
      GotoXY(5, 5); ClrEol; Write('Course. ', Course);
      GotoXY(5, 6); ClrEol; Write('User Status. ', Status);
      GotoXY(5, 7); ClrEol; Write('User Mode. ', Mode);
      GotoXY(5, 8); ClrEol; Write('User Password. ', Password);
      GotoXY(5, 9); ClrEol; Write('Last Used. ', Date);
      GotoXY(5, 10); ClrEol; Write('Last Login. ', Time);
      GotoXY(5, 11); ClrEol; Write('Progress. ', Progress);
      GotoXY(5, 12); ClrEol; Write('Grade. ', Grade);
      GotoXY(5, 13); ClrEol; Write('Grade point Avg. ', GradePoint);
      GotoXY(5, 14); ClrEol; Write('Times Used. ', TimesUsed);
      GotoXY(5, 15); ClrEol; Write('Press Any Key To Continue.. ');
      Ch := ReadKey;
      User := I;
      Inc(I);
    End;
  End;
End;
End;
Close (RecordFile);
End;

Procedure SearchForInfo;
Var UserIndex, Nindex : Integer;
Found, Nfound : Boolean;
Begin (SearchForInfo)
Found := False;
Nfound := False;
UserIndex := 0;
Repeat
With UserData[UserIndex] Do
Begin
DecideVar(Password, Length(Password));
SFpassword := Password;
SLastName := LastName;
EdUpcase(SFLastName);
EdUpcase(SFPassword);
EdUpcase(UserName);
EdUpcase(UserPassword);
If (SFLastName = UserName) Then
Begin
Nindex := UserIndex;
Nfound := True;
End;
If ((SFLastName = UserName)
AND (SFPassword = UserPassword)) Then
Begin
User := UserIndex;
MajorUser := User;
SelWindow(2);
Found := True;
Quit := True;
Str(User, UserString);
End;
UserIndex := UserIndex + 1;
Until (Found) or (UserIndex > MaxUser);
If Found Then UpdateInfo
Else
Begin (Record not found in file ROPE.LST)
If WrongPasswordTry < 3 then (Entered Wrong Password)
begin
Quit := False;
WrongPasswordTry := WrongPasswordTry + 1;
end
else
Begin
If (Nfound) Then
Begin
  CleanWindow(2);
  SelWindow(2);
  WriteLn('#7#77,' Do not try that one on me ');
  WriteLn(' [ ',UserPassword, ' ]');
  WriteLn(' is NOT a valid Password for you');
  WriteLn(' ',UserData[Nindex].FirstName, ',
           ',UserData[Nindex].LastName,'.' );
  WriteLn;WriteLn (' **** Try again ****');
  Halt(1);
End
else
Begin
  CleanWindow(2);
  SelWindow(2);
  WriteLn('#7#77.' Bye !! Bye !! ');
  WriteLn (' I Wonder Who You are. ' );
  WriteLn(' Do not try that one on me I Know');
  WriteLn(' [ ',UserPassword, ' ]');
  WriteLn(' is NOT a valid Password For you');
  WriteLn(' ',UserName );
  WriteLn (' You are NOT on the list ');
  WriteLn (' for this class/project.');
  Halt(0);
End;
End;

If (NOT FOUND) and ((Username <> #13) or (Username <> #0)) then
Begin
  CleanWindow(3);
  SelWindow(3);
  Write('#7#77.' Record For [',UserName,' ] Not Found Try Again');
  Delay(2500);
  CleanWindow(3);
End;
{ SearchForInfo }

Procedure MakeScreen;
Begin
  ClrScr;
  EdDrawBox(Border,1,1,80,3,NormalBox);
  EdFastWrite (CenterText(76, 'The following Information Must be provided to Run ROPE'),2,2,Yellow):

  DefineTextWindow(2,18,5,40,16);
  DefineTextWindow(3,5,21,70,3);
Procedure GetNameAndPword;
Var Ch: Char;
   PWord, NewPword: String;
   L, NLIIndex, LIndex: Integer;
Begin
   BkColor := Blue;
   ShowTextWindow(3);
  TextColor(Yellow);
   CleanWindow(3);
   SelWindow(3);
   If (Changes <> 'NEW') and (Changes <> 'OLD') then Begin
      GotoXY(5, 1);
      Write ('Enter Your Last Name');
      TextColor(White + Blink);
      Write (' --> ');
      TextColor(Yellow);
      Readin(UserName);
      CleanWindow(3);
      SelWindow(3);
     TextColor(Yellow);
      GotoXY(5, 1);
      Write ('Enter Your PassWord');
   End
   Else
   If changes = 'NEW' Then Begin
GotoXY(6,1);
Write('Enter New Password');
End
Else
If changes = 'OLD' Then
Begin
GotoXY(6,1);
Write('Enter Old Password');
End;
TextColor(White + Blink);
Write ('--> ');
TextColor(Yellow);
Lindex := 0;
Pword := ''; Repeat
If keypressed then
Begin
Ch := ReadKey;
If Ch <> #13 Then
Begin
PWord := Pword+ch;
LIndex := LIndex +1;
End;
End;
Until (ch = Chr(13)) ;
IF (ch = Chr(13)) and (Lindex = 0) then
Begin
Quit := True;
Username := #0;
End;
UserPassword := Pword;
If Keypressed then Ch := ReadKey;
End;

Procedure ListUsers;

Var I,SaveUser :Integer;

Chr :Char;
Begin
Assign (RecordFile,'ROPE.LST');
ReSet (RecordFile);
BkColor := Red;
SaveUser := User;
CleanWindow(2);
SelWindow(2);
I := 0;
While Not Eof(RecordFile) do
Begin
Seek(RecordFile,I);
Read(RecordFile, UserData[I]);
BkColor := Red;
CleanWindow(2);
SelWindow(2);
With UserData[I] do

Begin
  GotoXY(5, 2); Write('FirstName, ', 'LastName);
  GotoXY(5, 4); Write('Course.. ', 13, 'Course);
  GotoXY(5, 5); Write('User Status.. ', 13, 'Status);
  GotoXY(5, 6); Write('User Mode.. ', 13, 'Mode);
  GotoXY(5, 7); Write('Last Used.. ', 13, 'Date);
  GotoXY(5, 8); Write('Last Login.. ', 13, 'Time);
  GotoXY(5, 9); Write('Times Used.. ', 13, 'TimesUsed);
  CleanWindow(3);
  SelWindow(3);
  GotoXY(5, 1); Write('Press Any Key To Continue..');
  Chr := ReadKey;
End;
Inc(I);
End;
Close (RecordFile);
User := SaveUser;
End;

Procedure DeleteUser;
var I, J : Integer;
  SaveData : Boolean;
  TempUserData : Array [0..MaxUser] of UserDataType;
Begin
  Assign (RecordFile, 'ROPE.LST');
  ReSet (RecordFile);
  SaveData := False;
  BkColor := Red;
  CleanWindow(2);
  SelWindow(2);
  I := 0; J := 0;
  While Not Eof(RecordFile) do
    Begin
      Seek(RecordFile, I);
      Read(RecordFile, UserData[I]);
      BkColor := Red;
      CleanWindow(2);
      SelWindow(2);
      With UserData[I] do
        Begin
          GotoXY(5, 2); Write('FirstName, ', 'LastName);
          GotoXY(5, 4); Write('Course.. ', 13, 'Course);
          GotoXY(5, 5); Write('User Status.. ', 13, 'Status);
          GotoXY(5, 6); Write('User Mode.. ', 13, 'Mode);
          GotoXY(5, 7); Write('Last Used.. ', 13, 'Date);
          GotoXY(5, 8); Write('Last Login.. ', 13, 'Time');
GotoXY(5,8);Write('Last Login: 13,Time);
GotoXY(5,9);Write('Times Used: 13,TimesUsed);
CleanWindow(3);
SelWindow(3);
GotoXY(5,1);Write('Delete ( Y / N )');
Ch := Upcase(ReadKey);
Case Ch of
  'Y':
    Begin
      CleanWindow(3);
      SelWindow(3);
      GotoXY(2,1);
      Write('User ',FirstName,' ',LastName,
            ' is Deleted');
      Delay(1800);
    End;
  'N':
    Begin
      TempUserData[J] := UserData[I];
      Inc(J);
      SaveData := True;
    End;
End;
Inc(I);
End;
Close(RecordFile);
If SaveData then
  Begin
    {$1-}
    Assign (RecordFile, 'ROPE.BAK');
    Erase (RecordFile);
    Assign (RecordFile,'ROPE.LST');
    Rename (RecordFile,'ROPE.BAK');
    {$1+}
    Assign (RecordFile,'ROPE.LST');
    Rewrite (RecordFile);
    For I := 1 to J-1 do Write(RecordFile,TempUserData[I]);
    Close(RecordFile);
  End;
End;
Procedure ChangePassword;
Var OIdPassword :String;
Begin
  CleanWindow(3);
  SelWindow(3);
  Changes := 'OLD';
GetNameAndPword;
EdUpcase(UserPassword);
If (UserPassword = SFPassword) then
    Begin
        Changes := 'NEW';
        GetNameAndPword;
        EdUpcase(UserPassword);
        UserData[User].Password := UserPassword;
        CodeVar(UserData[User].Password,
                Length(UserData[User].Password));
        End
    Else
        Write(#7/#7,'Incorrect Old Password');
    End;

Procedure ChangeInfo;
var SaveUser,I,J,K :Integer;
    SaveData :Boolean;
    SGradePoint :String[4];
    TempUserData :Array [0..MaxUser] of UserDataType;
Begin
    SaveUser := User;
    Assign (RecordFile,'ROPE.LST');
    ReSet (RecordFile);
    SaveData := False;
    BkColor := * Red;
    CleanWindow(2);
    SelWindow(2);
    I := 0;J:=1;
    While Not Eof(RecordFile) do
        Begin
            Seek(RecordFile,I);
            Read(RecordFile,UserData[I+1]);
            Inc(I);
        End;
    Close (RecordFile);
    BkColor := * Red;
    CleanWindow(2);
    SelWindow(2);
    While J <= I-1 Do
        Begin
            With UserData[J] do
                Begin
                    User := J;
                    UpdateInfo;
                    CleanWindow(3);
                    SelWindow(3);
                End;
        End;
End;
GotoXY(5,1); Write('Edit Data ( Y / N )');
Ch := Upcase(ReadKey);
Case Ch of
  'Y':
    Begin
      ShowTextWindow(10);
      CleanWindow(10);
      SelWindow(10);
      GotoXY(2,16);
      WriteIn(' Enter New Data');
      WriteIn(' or');
      WriteIn('Press RETURN to Keep Old');
      CleanWindow(3);
      SelWindow(3);
      GotoXY(2,1);
      Write('Progress [ ',Progress,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Progress);
          Progress := Ch + Progress;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('G. P. A. [ ',GradePoint:7:3,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(SGradePoint);
          SGradePoint := Ch + SGradePoint;
          Val(SGradePoint,GradePoint,K);
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Grade [ ',Grade,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Grade);
          Grade := Ch + Grade;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Course [ ',Course,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Course);
          Course := Ch + Course;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Semester [ ',Semester,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Semester);
          Semester := Ch + Semester;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Year [ ',Year,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Year);
          Year := Ch + Year;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Year-List [ ',YearList,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(YearList);
          YearList := Ch + YearList;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Subject List [ ',SubjectList,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(SubjectList);
          SubjectList := Ch + SubjectList;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Degree Center [ ',DegreeCenter,'] --> ');
      Ch := Readkey;
      If Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(DegreeCenter);
          DegreeCenter := Ch + DegreeCenter;
          SaveData := True;
        end;
    end;
  else
    begin
      Ch := Readkey;
      if Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Progress);
          Progress := Ch + Progress;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('G. P. A. [ ',GradePoint:7:3,'] --> ');
      Ch := Readkey;
      if Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(SGradePoint);
          SGradePoint := Ch + SGradePoint;
          Val(SGradePoint,GradePoint,K);
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Grade [ ',Grade,'] --> ');
      Ch := Readkey;
      if Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Grade);
          Grade := Ch + Grade;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
      Write('Course [ ',Course,'] --> ');
      Ch := Readkey;
      if Ch <> RETURN then
        begin
          Write(Ch);
          ReadIn(Course);
          Course := Ch + Course;
          SaveData := True;
        end;
      GotoXY(2,1);ClrEol;
    end;
Write(Ch);
Readln(Course);
Course := Ch + Course;
SaveData := True;
end;
GotoXY(2,1);ClrEol;
Write ('Status [ ',Status,' ] --> ');
Ch := Readkey;
If Ch <> RETURN then
begin
Write(Ch);
Readln(Status);
Status := Ch + Status;
SaveData := True;
end;
GotoXY(2,1);ClrEol;
Write ('Mode [ ',Mode,' ] --> ');
Ch := Readkey;
If Ch <> RETURN then
begin
Write(Ch);
Readln(Mode);
Mode := Ch + Mode;
SaveData := True;
end;
GotoXY(2,1);ClrEol;
DeCodeVar(Password.Length(Password));
Write ('Password [ ',Password,' ] --> ');
Ch := Readkey;
If Ch <> RETURN then
begin
ChangePassword;
SaveData := True;
end;
End;
'N':SaveData := False;
End;
TempUserData[j] := UserData[j];
Inc(j);
End;
If SaveData then
Begin
Assign (RecordFile,'ROPE.LST');
Rewrite (RecordFile);
For I := 1 to J-1 do
Write(RecordFile,UserData[I]);
Close(RecordFile);
End;
User := SaveUser;
End;

Procedure AddUser;
Begin
Assign (RecordFile2, 'ROPE.LST');
Reset (RecordFile2);
SetTextWindow(2);
CleanWindow(2);
SelWindow(2);
WriteIn(' Enter Data For Each User ');
WriteIn(' Press ESC when Done ');
EndData := False;
CleanWindow(3);
SelWindow(3);
While Not EndData do
Begin
GetRecordInfo;
If Not EndData then
Begin
Seek(RecordFile2,FileSize(RecordFile2));
Write (RecordFile2, NewUser);
End;
End;
Close (RecordFile2);
End;

Procedure GetUserStatus;
Var WIndex, I, J: Integer;
    Ch, Ch1, Ch2 : Char;
Begin
Assign (RecordFile, 'ROPE.LST');
Reset (RecordFile);
DefineTextWindow(10, 52, 7, 27, 8);
Profile := False;
I := 0;
HelpIndex := 2;
User := 0;
Quit := False;
MakeScreen;
While Not Eof(RecordFile) do
Begin
Seek(RecordFile, I);
Read(RecordFile, UserData[I+1]);
Inc(I);
End;
Close(RecordFile);
Assign (RecordFile, 'ROPEPF.LST');
Reset (RecordFile);
J := 0;
While Not Eof(RecordFile) do
  Begin
    Seek(RecordFile, J);
    Read(RecordFile, UserData[I]);
    Inc(J);
    Inc(I);
  End;
Close (RecordFile);
Bkcolor := 0;
WrongPasswordTry := 1;
Changes := ' NO';
Repeat
  GetNameAndPword;
  SearchForInfo;
Until Quit;
Repeat
  UpdateInfo;
  TextBackground(BkColor);
  CleanWindow(3);
  Window(1,1,80,25);
  Quit := False;
  CleanWindow(3);
  SelWindow(3);
  If ((UserData[User].Status = 'Student') or
      (UserData[User].Status = 'Graduate')) and
      (UserData[User].Mode = 'Protected') then
    Begin
      TextColor(White);
      GotoXY(6,1);
      Write ('F1 Help/F7 Change Password /',
             'Any other Key To Cont...');
      Ch := ReadKey;
      If Ch = #0 then
        Begin
          Ch := ReadKey;
          If Ch = #65 then ChangePassword;
          If Ch = #59 then GetHelp;
        End
      Else Ch := 'Q';
    End
  Else
    If ((UserData[User].Status = 'Student') or
      (UserData[User].Status = 'Graduate')) and
      (UserData[User].Mode = 'Unprotected') then
    Begin
      Bkcolor := 1;
      CleanWindow(10);
      ShowTextWindow(10);
      SelWindow(10);
GotoXY(2,2);Write('F1-Make Student List');
GotoXY(2,3);Write('F2-See List');
GotoXY(2,5);Write('F9-To Continue..');
Ch1 := ReadKey;
If Ch1 = #0 then
Begin
  Ch1 := ReadKey;
  Case Ch1 of
    #59: NameRecords;
    #60: ListUsers;
    #67: Ch1 := '0'
      Else Write (#7);
  End;
End
Else
Write(#7#7);
End
Else
If (UserData[User].Status = 'Professor') or
(UserData[User].Status = 'Programmer') then
Begin
  Bkcolor := 1;
  CleanWindow(10);
  ShowTextWindow(10);
  SelWindow(10);
  GotoXY(2,1);Write('F1-Make Student List');
  GotoXY(2,2);Write('F2-See List');
  GotoXY(2,3);Write('F3-Edit list');
  GotoXY(2,4);Write('F7-Change Password');
  GotoXY(2,6);Write('F9-Continue..');
  Ch1 := ReadKey;
  If Ch1 = #0 then
  Begin
    Ch1 := Readkey;
    Case Ch1 of
      #59: NameRecords;
      #60: ListUsers;
      #61:
        Begin
          CleanWindow(10);
          ShowTextWindow(10);
          SelWindow(10);
          GotoXY(2,1);Write('F2-See List');
          GotoXY(2,2);Write('F4-Delete User');
          GotoXY(2,3);Write('F5-Change Info.');
          GotoXY(2,4);Write('F6-Add User');
          GotoXY(2,6);Write('F10-Enable Quit');
          Ch2 := ReadKey;
          If Ch2 = #0 Then
            Begin
Ch2 := ReadKey;
Case Ch2 of
  #59:GetHelp;
  #60:ListUsers;
  #62:DeleteUser;
  #63:ChangeInfo;
  #64:AddUser;
  #65:ChangePassword;
  #68:
    Begin
      Quit := True;
      QuitProgram := True;
      End
    Else Write(#7#7);
    End;
  End
Else
  Write(#7#7);
End;
Else
  Begin
    CleanWindow(3);
    SelWindow(3);
    Write(#7#7,'Error in your Status or Mode Check your Instructor');
    Halt(4);
  End;
Until Ch = 'Q';
ExitRecAndUpdate;
BkColor := 0;
End;

End. { unit }
Unit SPC_Help:

Interface

Uses
  Crt.Dos.
  Spc_Id.
  Turbo3.
  GDVner.
  Introd:

Type
  HelpArray = Array [0..255] of HelpRec:
  StringArray = Array [0..50] of String[80]:
  HelpRec = Record
    IdString: String[50];
    Info : StringArray;
    Lines : Integer;
  End;

Var
  HelpInfo : HelpRec;
  k ,k : Integer;
  HelpFile : File of HelpRec;
  Ch : Char;
  InfoFile : Text;
  Buffer : Word;
  Helps : HelpArray;

Procedure GetHelp:

{=======================================================}

Implementation

Procedure GetHelp:
Label HelpRedo, HelpBegin, HelpEnd:
begin { Get Help }
  HelpBegin:
    HighVideo:
    {$I-}
    Assign (HelpFile, 'Help.Hlp');
    SetAttr(HelpFile, Archive);
    ReSet (HelpFile):
    {$I+}
  If IOResult <> 0 then
    Begin
      Write(#7#7, '*** Error Reading Help File *** Aborting Program ***');
      Halt(10);
DefineTextWindow(1,2,2,75,22);
ShowTextWindow(1);
I := HelpIndex;
Ch := #0;
While Not Eof(HelpFile) do Begin
  CleanWindow(1);
  Seek(HelpFile, I);
  Read(HelpFile, HelpInfo);
  With HelpInfo Do Begin
    GotoXY(2,(75-Length(IdString) div 2));
    Write(IdString);
    For k := 0 to Lines do Begin
      If Lines < 9 then GotoXY (2,4+(2*k))
      Else GotoXY (2,1+k);
      Write(Info[k]);
    End;
  End;
End;
HelpRedo:
TextBackground(red);
GotoXY (2,20);
Write ('Press Esc to leave Help / Use --Home PgUp PgDn End ' 
      ,#24#25,' To Scroll');
TextBackground(Blue);
Ch := Readkey;
Case Ch of
  #27:  Goto HelpEnd;
  #0:
    Begin
      Ch := ReadKey;
      Case Ch of
        #71:I := 0;
        #72,#73:If I > 0 then I := I-1;
        #79:I := FileSize(HelpFile)-1;
        #80,#81: Inc(I)
      End;
    End
  Else
    Begin
      Goto HelpRedo;
      Write(#7#7);
    End;
End;
End;
Close (HelpFile);
SetFAttr(HelpFile,ReadOnly);
Goto HelpBegin:
HelpEnd:
End:

end. { Get Help }
Unit RopeMs:

{Full blown editor using the Turbo Editor Toolbox}

Interface

{$I msdirect.inc}
{$L-

uses
Crt.
Dos.
Errors.
MsVars.
MsScr1.
MsString.
MsPtrOp.
EscSeq.
MsCmds.
Int24.
Message.
MsUser.
MsBack.
MsScr2.
MsMenu.
MsDir.
MsEdit.
MsText.
MsTabs.
MsBlok.
MsFind.
MsFile.
MsMacro.
MsSet.
MsPrtM.
MsSpell.
Invoke.
MsInvoke.
MsMain;

Procedure RopeEdt:

Implementation

Procedure RopeEdt:

begin

{MicroStar-ROPE Editor}

{Initialize toolbox data structures}
EdInitialize:

{Set up for the first file}
EdMainMenu:

{Start the keyboard polling loop}
EdSchedule:

{Restore the screen}
EdRestoreScreenMode:
end; {MicroStar-ROPE Editor}

End. {Unit}
Unit SPC_Menu:

Interface

Uses
  Crt,
  Dos,
  RopeMs,
  MsScrn1,
  MsUser,
  MsString,
  Turbo3,
  SPC_Id,
  SPC_Util,
  Intro,
  Invoke,
  SPC_Cmd,
  SPC_KAL,
  SPC_Help,
  SPC_Grap;

var
  Q,R:WindowPtr;
  Code,I :Integer;
  CleanChar:Char;
  Prompt2,PromptLine:String;
  HeapPtr :Integer;

Procedure CleanVideoMemory;

Procedure PrintErrorMsg(Code:Integer);

Procedure DrawMainMenu;

Procedure SaveMainWindow;

Procedure RestoreMainWindow;

Procedure DoDosShellProcess;

Procedure GetMenuChoice;

Procedure MainMenu;

{===============================================================================}

Implementation
Procedure CleanVideoMemory;
Var I: Integer;
begin
  TextBackground(black);
  I := 0;
  repeat
    Move(CleanChar, Mem[$8000:I], SizeOf(CleanChar));
    I := I + 2;
  until I >= 4000;
end;

Procedure SaveMainWindow;
Begin
  Q := EdSetupWindow(Border, 22, 8, 57, 19, NormalBox);
End;

Procedure RestoreMainWindow;
Begin
  EdRestoreWindow(Q, 22, 8, 36, 12);
End;

Procedure PrintErrorMsg(Code: Integer);
Begin
  SaveMainWindow;
  DefineTextWindow(5, 22, 8, 37, 14);
  { ShowTextWindow(5); }
  SelWindow(5);
  GotoXY(1, 1);
  Write(#7#7);
  GotoXY(3, 3);
  Case Code of
    -1 : Writeln(' Insufficient memory to store free list ');
    -2 : Writeln(' DOS setblock error before EXEC call ');
    -3 : Writeln(' DOS setblock error after EXEC call -- critical error ');
    -4 : Writeln(' Insufficient memory to run DOS command ');
    0  : Writeln(' *** No Such Choice ');
    8  : Writeln(' Not Enough Memory ');
  End;
  Delay(2000);
  DefineTextWindow(5, 1, 1, 80, 25);
  SelWindow(5);
  RestoreMainWindow;
End;

Procedure DrawMainMenu;
Begin
  EdsetCursorOff;
EdFastWrite(CenterText(76,'WELCOME TO ROPE'),4,2,11);
EdFastWrite(CenterText(76,'MAIN MENU'),9,2,12);
EdFastWrite(CenterText(76,'Please Enter Your Selection ':'),11,2,14);

EdFastWrite(CenterText(76,'EdPadEntry('F1 Help'),13,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F2 Statistics'),14,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F3 Calibrations'),15,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F4 Editor'),16,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F5 Shell to DOS'),17,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F6 Exit'),18,2,2);
EdFastWrite(CenterText(76,'EdPadEntry('F10 ROPE'),19,2,2);

EdBox(Border,1,1,80,25,ErrorBox);
End;

Procedure DoDosShellProcess;
Begin
  Mark(HeapPtr);
  HelpIndex := 10;
  R:= EdSetupWindow(Border,1,1,80,25,NormalBox);
  DefineTextWindow(1,3,3,77,23);
  ShowTextWindow(1);
  SelWindow(1);
  TextBackground(Black);
  ClrScr;
  GotoXY(1,1);
  Repeat
    window(2,24,79,25);
    HighVideo;
    Textcolor(White);
    TextBackground(Blue);
    GotoXY(1,1);
    Write(' ------> Type Rope to Return to ROPE <------ ');
    GotoXY(1,2);ClrEol;GetDir(0,Prompt2);
    Write(Prompt2,' :ROPE> ');
    GotoXY(WhereX+1,WhereY);
    Readln(PromptLine);
    EdUpcase(PromptLine);
    TextBackground(Black);
    EdSetCursor($0607);
    If PromptLine <> 'ROPE' then
      Begin
        SelWindow(1);
        ClrScr;
        GotoXY(3,1);
        Code := ExecSHrink(PromptLine);
        If DosError <> 0 Then PrintErrorMsg(DosError);
      End;
      ...
Until PromptLine = 'ROPE';
NormVideo;
EdRestoreWindow(R,1,1,80,25);
Release(HeapPtr);
End;

Procedure GetMenuChoice;
Var
  Ch:Char;
Begin
  Quit := False;
  Repeat
    EdSetCursorOff;
    Out := False;
    Ch := ReadKey;
    If (Ch = #0) Then
      Begin
        Ch := ReadKey;
        Case Ch of
          #59:
            Begin
              Mark(HeapPtr);
              R := EdSetupWindow(Border,1,1,80,25,NormalBox);
              HelpIndex := 5;
              TextBackground(Black);
              ClrScr;
              GotoXY(5,5);
              GetHelp;
              EdSetCursorOff;
              EdRestoreWindow(R,1,1,80,25);
              Release(HeapPtr);
              End;
          #60:
            Begin
              Mark(HeapPtr);
              R := EdSetupWindow(Border,1,1,80,25,NormalBox);
              HelpIndex := 6;
              TextBackground(Black);
              ClrScr;
              GotoXY(5,5);
              Write ('This Function is to be Added');
              GotoXY(5,7);
              Write('Press a key when ready');
              Delay(25000);
              EdSetCursorOff;
              EdRestoreWindow(R,1,1,80,25);
              Release(HeapPtr);
              End;
          #61:
  End;
Begin
  Mark(HeapPtr);
  R := EdSetupWindow(Border, 1, 1, 80, 25, NormalBox);
  DefineTextWindow(1, 1, 80, 25);
  SelWindow(1);
  TextBackground(Black);
  ClrScr;
  GotoXY(5, 5);
  RegresXY;
  EdSetCursorOff;
  EdRestoreWindow(R, 1, 1, 80, 25);
  Release(HeapPtr);
End;

#62:
Begin
  Mark(HeapPtr);
  R := EdSetupWindow(Border, 1, 1, 80, 25, NormalBox);
  DefineTextWindow(1, 1, 80, 25);
  SelWindow(1);
  TextBackground(Black);
  ClrScr;
  GotoXY(5, 5);
  HelpIndex := 9;
  RopeEdit;
  EdSetCursorOff;
  EdRestoreWindow(R, 1, 1, 80, 25);
  Release(HeapPtr);
End;

#63: DoDOSShellProcess:

#64:
Begin
  HelpIndex := 11;
  Quit := True;
End;

#68:
Begin
  Mark(HeapPtr);
  R := EdSetupWindow(Border, 1, 1, 80, 25, NormalBox);
  DefineTextWindow(1, 1, 80, 25);
  CleanWindow(1);
  SelWindow(1);
  TextBackground(Black);
  ClrScr;
  GotoXY(5, 5);
  EdSetCursorOff;
  HelpIndex := 7;
  ProcessIntro;
  DoRPE;
  EdSetCursorOff;
  EdRestoreWindow(R, 1, 1, 80, 25);
Release(HeapPtr);
End;
End;
End
Else
Begin
Mark(HeapPtr);
EdSetCursorOff;
PrintErrorMsg(0);
EdSetCursorOff;
Release(HeapPtr);
End;
Until Quit:
End:

Procedure MainMenu:
Begin
CleanChar := #0;
ClrScr;
DrawMainMenu;
GetMenuChoice;
End;
End. { unit }

Program SPC_ROPE:

Uses
Crt.Dos, (General units for display and use of Disk Operating System (DOS))
Turbo3, ( Unit to convert Version 3.0 codes )
INTROD, ( Unit for showing Introduction )
SPC_Tit, ( Unit to show title page )
SPC_ID,
SPC_REC, ( Unit to get user information and file management )
SPC_MENU
;
Var MainPtr :Integer;
CH :Char;

begin { main program }
Mark(MainPtr);
QuitProgram := False;
Repeat
  Title;
  GetUserStatus;
  ClrScr;
  Introduction;
  Release(MainPtr);
  Mark(MainPtr);
  MainMenu;
  until QuitProgram;
  Clrscr;
  Release(MainPtr);
end. { main program }
APPENDIX B. INTERFACING SOFTWARE PROGRAM LISTINGS
(*-------------------------------------------------------------*)
(* Statistical Process Control package                           *)
(* By Renford A. B. Brevett                                    *)
(* Module version 1.03A                                         *)
(* Copyright (c) 1988 by Iowa State University Ames, Ia. 50010  *)
(*                                                            *)
(*-------------------------------------------------------------*)

Unit SPC_FLOWS:

Interface

Uses

  Crt,
  Turbo3,
  SPC_ID,
  SPC_UTIL,
  Introd,
  SPC_tit,
  GKernel,
  GDriver;

  Var ch :Char;
    TextSize :Integer;

Procedure BinSet:

Procedure NightFlow:  ( Set the system to flow between the storage
                       and the house. Energy stored will be extracted
                       from the storage unit and use to keep the house
                       at the specified temperature. The downdrain valve
                       is first closed. )

Procedure NormalFlow:  ( Set the system to flow between the pannel,
                        the house and resovoir. )

Procedure StorageFlow: ( Set the system to flow between the pannel, storage,
                        the house, and resovoir. Energy will be stored in the
                        the storage unit and will be use to keep the house
                        at the specified temperature when needed. The downdrain valve
                        is first closed. )

Procedure CoolingSystemFlow: { Set the system to flow through the cooling pipes and radiator to cool pannel and/or liquid. The downdrain valve is first closed. }

Procedure DowndrainMain: { Set the system to drain all the liquid into the reservoir. The downdrain valve is kept for a period of time to allow maximum drainage. }

Procedure DowndrainAuxillary: { Set the system to drain all the liquid into the reservoir. The downdrain valve is kept for a period of time to allow maximum drainage. }

Procedure TestProcess;  

{===============================================================}

Implementation

Procedure BinSet;  
Begin  
IniDevice(CardNumber,BString);  
End;

Procedure NightFlow: { Set the system to flow between the storage and the house. Energy stored will be extracted from the storage unit and used to keep the house at the specified temperature. The downdrain valve is first closed. }

Begin  
BinSet;

GotoXY(2,2);  
IF FirstTime Then  
If GrafModeGlb then  
    DrawTextW(2,6,TextSize,'Testing Night Flow')  
Else  
    Begin  
        ClearEol(21);  
        Write ('Testing Night Flow');  
    End;

TurnSwitchOff (4); { Close DownDrain valve }
WriteBinaryData (CardNumber, NightFlowCode); { Turn Secondary Pump on and set Switch 1 & 3 to allow flow between house and storage. }
Procedure NormalFlow;  { Set the system to flow between the panel,  
the house and reservoir.  }

Begin
  GotoXY(2.2):
  BinSet:
  If FirstTime then
    If GrafModeGlb then
      DrawTextW(2.6,TextSize,'Testing Normal Flow')
    Else
      Begin
        ClearEol(21);
        Write ('Testing Normal Flow');
      End;
  End;
  TurnSwitchOff (4);  { Close DownDrain valve }
  WriteBinaryData (CardNumber, NormalFlowCode );  { Turn Primary Pump on and  
set Switch 1, 2 & 3 to allow flow between  
house, panel and reservoir. The  
downdrain valve is first closed.}
End;  { NormalFlow }  

Procedure StorageFlow;  { Set the system to flow between the panel, storage,  
the house, and reservoir. Energy will be stored in the  
the storage unit and will be used to keep the house  
at the specified temperature when needed. The downdrain valve  
is first closed. }

Begin
  GotoXY(55,21):
  BinSet:
  If FirstTime then
    If GrafModeGlb then
      DrawTextW(2.6,TextSize,'Testing Storage Flow')
    Else
      Begin
        ClearEol(21);
        Write ('Testing Storage Flow');
      End;
  End;
  TurnSwitchOff (4);  { Close DownDrain valve }
  WriteBinaryData (CardNumber, StorageFlowCode );  { Set Switch 2 to allow flow  
through storage. }
End;  { StorageFlow }
Procedure CoolingSystemFlow: (Set the system to flow through the cooling pipes and radiator to cool panel and/or liquid. The downdrain valve is first closed.)

Begin
  GotoXY(55,21);
  BinSet;
  If FirstTime then
    If GrafModeGlb then
      DrawTextW(2,6,TextSize,'Testing Cooling System Flow')
    Else
      Begin
        ClearEol(21):
        Write ('Testing Cooling system Flow');
      End;
  EndSwitchOff (4); (Close DownDrain valve)
  WriteBinaryOata (CardNumber, CoolingSystemCode ); (Turn Secondary Pump on and set Switch 1 & 3 to allow flow between house and storage.)
End; (CoolingSystemFlow)

Procedure DowndrainMain: (Set the system to drain all the liquid into the reservoir. The downdrain valve is kept for a period of time to allow maximum drainage.)

Begin
  BinSet;
  WriteBinaryOata (CardNumber, DowndrainMainCode ); (Set switch 3 & 4 on to allow air into system.)
End; (DowndrainMain)

Procedure DowndrainAxillary: (Set the system to drain all the liquid into the reservoir. The downdrain valve is kept for a period of time to allow maximum drainage.)

Begin
  BinSet;
  WriteBinaryOata (CardNumber, DowndrainAxillaryCode ); (Set switch 2 & 4 on to allow air into system. This allow Secondary sections to be drained.)
End; (Downdrainaxillary)

Procedure TestProcess;
Begin
  TextSize := 1;
  SelectWorld(4);
  SelectWindow(7);
  SetBackground(0);
  NormalFlow;
  WaitTime(5);
  SelectWindow(7);
  SetBackground(0);
  StorageFlow;
  WaitTime(5);
  SelectWindow(7);
  SetBackground(0);
  NightFlow;
  WaitTime(5);
  SelectWindow(7);
  SetBackground(0);
  CoolingSystemFlow;
  WaitTime(5);
  GotoXY(55,22);
  SelectWindow(7);
  SetBackground(0);
  If GrafModeGlb then
    DrawTextW(2,5,TextSize,'End of Test')
  Else
    Writeln('*** End of Test ***');
  GotoXY(55,23);
  If GrafModeGlb then
    DrawTextW(2,7,TextSize,'<<Press Any Key To Continue>>')
  Else
    Writeln('<<Press any Key to continue...>>');
  WriteBinaryData(CardNumber,0);
  repeat Until Keypressed;
  SelectWindow(7);
  SetBackground(0);
  Ch := ReadKey;
end;

End. (unit)
Unit SPC_Proc:

Interface

Uses
  Crt,
  Turbo3,
  SPC_Id,
  SPC_Util1,
  SPC_Floa,
  MsUser,
  MsScrni,
  Introd,
  SPC_TIT,
  GDri,er,
  GKernel,
  GWindow,
  GShell;

Const
  ESC = $1b;

Var
  WP :WindowRec;
  Ch :Char;

Procedure StatusCheck;

Procedure SystemCheck;

Procedure StartProcess;

Procedure ControlProcess;

implementation

Procedure StatusCheck;

Var ADStatus :String;
  TextSize :Integer;
Begin
  SelectWorld(4);
  SelectWindow(7);
  TextSize := 1;
  SetBackground(0);
  Data := ReadBinaryData(CardNumber);
Gotoxy(55,20);
If TestBit(Data.7) then
  ADStatus := 'Power Is On'
Else
  ADStatus := 'Power Is Off';
If GrafModeGlbl then
  DrawTextW(3.3,TextSize,ADStatus)
Else
  Write (ADStatus);
GotoXY(55,21);
If TestBit(Data.4) Then
  begin
    ADStatus := 'Downdrain valve is OPEN' + '#13#10+tab5' + 'It will be Closed Automatically';
    TurnSwitchOff(4);
  end
Else
  ADStatus := 'Downdrain valve is CLOSED';
If GrafModeGlbl then
  DrawTextW(3.5,TextSize,ADStatus)
Else
  Write (ADStatus);
WaitTime(3);
GotoXY(55,21);
If TestBit(Data.8) Then
  ADStatus := 'A/D converter switch is ON'
else
  ADStatus := 'A/D Converter Switch is OFF';
If GrafModeGlbl then
  DrawTextW(2,7,TextSize,ADStatus)
Else
  Write (ADStatus);
End;

Procedure SystemCheck;
Begin
  GetTemperature;
End;

Procedure StartProcess;
Begin
  SelectWorld(4);
  SelectWindow(7);
  ResetSystem;
  StatusCheck; { Check for Power on, closed air inlet valve and
  Check for Switch power at D/A converter. }
  WaitTime(3);
  TurnSwitchOn(5); { Turn power on to the system }
  StatusCheck; { Check for Power on, closed air inlet valve and
  Check for Switch power at D/A converter. }
TestProcess;
PrepareLogFile;
FirstTime := False;
NormalFlow;
WaitTime(20); {Time to let system stabilize}
Quit := False;
ShutDown := False;
SystemStartTime := FetchTime;
CycleTime := 0;
End;

Procedure ControlProcess;

Begin
CycleTime := Abs(FetchTime-SystemStartTme);
If (Not ShutDown) and (CycleTime >= 10) and (CycleTime < 30) then NormalFlow;
SystemCheck: {Check for the controlling variables for out of control conditions and update screen}
AdjustSystem; {Make adjustment to system when necessary}
If ShutDown then
Begin
{SI-}
LogData;
Close (DataFile);
{SI+}
WaitTime(2);
If Keypressed then Ch := Upcase(Readkey);
If Ch = '0' Then Exitgrap := True;
SystemCheck;
AdjustSystem;
End
Else:
Begin
If (Not ShutDown) and (CycleTime = 30) And (CycleTime < 60) then
StorageFlow;
If (Not ShutDown) and (CycleTime = 60) And (CycleTime < 100) then
CoolingSystemFlow;
If (Not ShutDown) and (CycleTime >= 100) then
Begin
CycleTime := 0;
SystemStartTime := FetchTime;
End;
End;
End;
End. { unit }
APPENDIX C. DATA DISPLAY AND ANALYSIS PROGRAM

LISTINGS
Unit SPC_Grap:

Interface

Uses
Dos.crt.Printer.turbo3,
GDriver,Gkernel,Gwindow,Gshell,Spc_Id,
IntroSpeg_Util,SPC_tit,SPC_Proc,
SPC_Help,MsUser;

Const
MaxSelWin = 6;
MinSelWin = 2;

Var
SelWin,I :Integer;
ZoomWorld :Integer;
ZoomWin :Array[MinSelWin..MaxSelWin] of
  Boolean;
m,n,o,p,q :integer;
a,b,c,d,e :PlotArray;
TxtPtr :Integer;
XBar,UCL,LCL :Real;

procedure SetUpGrapWindows;
Procedure SelGrapWindow;
procedure UpDateGraph;
Procedure Info7;
Procedure PlotA11Graph;
Procedure CheckForInput;
procedure WindowControlAndZoom;
Procedure SavePicOrTxtGrapScreen;

Procedure DoROPE:

{------------------------------------------------------------------}
Implementation

procedure SetUpGrapWindows;

var
    FColor, WinSelect, q: integer;
    WlnlName: String;

Begin
    FColor := 7;
    WinSelect := 1;
    q := 1;
    HelpIndex := 3;
    SetForegroundColor(FColor);
    defineWindowSPC(1,0,0,79,199);
    defineWindowSPC(8,0,0,79,30);
    defineWindowSPC(9,0,174,79,198);
    defineheader(1,'ROPE Charting System');
    WlnlName := 'ROPE:Statistical Process Control Tool';
    WlnlName := 'ROPE:Statistical Process Control Tool'
        + ' by R.A.B. Brevett at Iowa State University.';
    defineheader(8, WlnlName);
    defineheader(9, 'Use Function Keys to Change or View Graphs');
    setHeaderOff;
    SelectWindow(1);
    DrawBorder;
    SetHeaderOff;
    SelectWindow(9);
    DrawBorder;
    SetHeaderOn;
    Delay(11000);
    SelectWindow(8);
    DrawBorder;
    clearEol(2);
    gotoxy(2,3);
    Write ('F1-Help F2-Select Window F3-Zoom');
    Write (' F4-Print F5-Save F6-Change Variables F10-Exit');
    Delay(800);
    setForegroundColor(FColor);
end;

{------------------------------------------------------------------}
Procedure SelGrapWindow;

Var Ch : Char;

Begin
    SelectWindow(8);
DrawBorder;
SetHeaderOFF;
ClearEol(2);
GotoXY(12.2);
Write ('Use '.,#24#25#26#27.,' to Select Window and ESC when Done');
SelWin := 2;
SetHeaderON;
SelectWindow(SelWin);
InvertWindow;
Beep2:
Repeat
  Ch := ReadKey;
  IF Ch = #0 Then
    Begin
      Ch := ReadKey;
      Case Ch of
        #71:
          Begin
            InvertWindow;
            SelWin := MinSelWin;
            SelectWindow(SelWin);
            End;
        #72:
          Begin
            InvertWindow;
            If (SelWin > 4) Then SelWin := SelWin - 3
            Else Beep;
            SelectWindow(SelWin);
            End;
        #73:
          Begin
            InvertWindow;
            If (SelWin > 4) Then SelWin := SelWin - 3
            Else Beep;
            SelectWindow(SelWin);
            End;
        #75:
          Begin
            InvertWindow;
            If (SelWin > MinSelWin ) Then SelWin := SelWin - 1
            Else Beep;
            SelectWindow(SelWin);
            End;
        #77:
          Begin
            InvertWindow;
            If (SelWin < MaxSelWin) Then SelWin := SelWin + 1
            Else Beep;
            SelectWindow(SelWin);
            End;
#79:  
Begin  
InvertWindow:  
SelWin := MaxSelWin;  
SelectWindow(SelWin):  
End;  

#80: Begin  
InvertWindow:  
If (SelWin < 5) Then SelWin := SelWin + 3  
Else Beep;  
SelectWindow(SelWin):  
End;  

#81: Begin  
InvertWindow:  
If (SelWin < 5) Then SelWin := SelWin + 3  
Else Beep;  
SelectWindow(SelWin):  
End;  
End;  
InvertWindow:  
End;  
Until Ch = #27;  
ClearEol(2);  
End;  

procedure UpDateGraph;  
var  
  x1,x2,i: integer;  

procedure RBGenerateFunction(var a,b:PlotArray;n:integer):  
var i:integer;  
  delta:real;  
begin  
  delta:=2*pi/(n-1);  
  for i:=1 to n do  
    begin  
      a[i,1]:=(i-1)*delta*pi;  
      b[i,1]:=a[i,1];  
      b[i,2]:=exp(-abs(a[i,1]));  
      a[i,2]:=sin(a[i,1])*exp(-abs(a[i,1])*0.1);  
    end;  
end;  

begin  
  SetColorWhite;  
  n:=30;  
  RBGenerateFunction(a,b,n);  
  { DefineWorld(2,-2,-1.2,1.2);  

{ DefineWorld(4.3*pi/2, -1.3*pi/2, 1); }
{ SelectWorld(q); }
end:

(---------------------------------------------------------)

Procedure DrawGraphAxes;
Const
MaxWorldX :Float = 100.0;
MaxWorldY :Float = 100.0;
Var Col, Row :Integer;
CharH, CharW :Float;
Begin
SelectWorld(1);
CharH := MaxWorldY / 25;
CharW := MaxWorldX / 80;
SetLineStyle(0);
DrawAxis(7.-7.0,0.0,0.0,0.-0,false);
End:

(---------------------------------------------------------)

Procedure DrawControlLimits;
Var Row :Integer;
Begin
SelectWorld(1);
SetLineStyle(0);
SelectWorld(1);
Drawline(X1RefGlb+20,X1Glb+UCL,X2RefGlb+80,X1WldGlb+UCL);
ResetAxis;
SelectWorld(1);
Drawline(X1RefGlb+20,X1Glb+LCL,X2RefGlb+80,X1WldGlb+LCL);
ResetAxis;
SelectWorld(1);
Drawline(X1RefGlb+20,X1Glb+XBar,X2RefGlb+80,X1WldGlb+XBar);
SelectWorld(1);
ResetAxis;
{ DrawText(X2RefGlb+85,X1Glb+Int(UCL),1,‘UCL’);
SelectWorld(1);
ResetAxis;
DrawText(X2RefGlb+85,X1Glb+LCL,1,‘LCL’);
SelectWorld(1);
ResetAxis;
DrawText(X2RefGlb+85,X1Glb+XBar-1.1,‘_’);
SelectWorld(1);
ResetAxis;
DrawText(X2RefGlb+85,X1Glb+XBar,1,‘X’);
SelectWorld(1);
ResetAxis;
GotoXY(WindowX((X2RefGlb+)80),WindowY((X1WldGlb+)XBar));
Write(‘UCL-Real’);
ResetAxis;
DrawText(X1Glb+60,60,1,'_');
DrawText(X2Glb-20,60,1,'R');
DrawText(X2Glb,UCL,1,'UCL');
DrawText(X1Glb+10,LCL,1,'LCL');
DrawText(X2Glb-10,XBar,1,');
DrawText(X1Glb-10,XBar,1,'X');
DrawText(X1Glb+10,60,1,'_');
DrawText(X2Glb+10,60,1,'R');
End;

Procedure XBarGraph2;

var 1:Integer;
Y:Float;
s:strings;
ch:char;

Begin
Randomize;
defineWindow(2,1.33,25,99);
'defineheader(2,'X-Bar Chart'+tab5+'Window<2>');</
If ZoomWin[2] then RedefineWindow(2,1,1,75,180);
SelectWorld(id);
SelectWindow(2);
SetBackground(0);
DrawBorder;
DrawGraphAxis;
for i:=1 to NumOfData do
begin
  c[i,1]:=i;
  c[i,2]:=SampleArray[i];
  Str(SampleArray[i],s);
  DrawText(50,i*5,1,s+' ,');
end;
SelectSPC;
DrawControlLimits;
ResetAxis;
SelectWorld(1);
Drawline (X1Glb+20,X1Glb+UCL,X2RefGlb+80,X1WldGlb+UCL);
ResetAxis;
SelectWorld(1);
Drawline (X1Glb+20,X1Glb+LCL,X2RefGlb+80,X1WldGlb+LCL);
ResetAxis;
SelectWorld(1);
Drawline (X1Glb+20,X1Glb+XBar,X2RefGlb+80,X1WldGlb+XBar);
SelectWorld(1);
ResetAxis;
DrawPolygon(c,1,NumOfData,-9,2.0);
ResetAxis;
End;

Procedure ParetoGraph3;
Var i : Integer;
Begin
  defineWindowSPC(3,27,33,51,99);
  defineHeader(3,'Pareto Analysis'+tab5+'Window<3>'); 
  If ZoomWin[3] then redefineWindow(3,1,1,75,180);
  SelectWorld(1);
  SelectWindow(3);
  SetBackground(0);
  DrawBorder;
  DrawGraphAxis;
    for i:=1 to NumOfData do 
    begin
      c[i,1]:=i;
      c[i,2]:= SampleArray[i];
    end;
  XBar := SampleArray[50];
  UCL := SampleArray[75];
  LCL := 0;
  { DrawGraphAxis; }
  DrawPolygon(c,1,NumOfData,-9.2,0);
  ResetAxis;
  Spline(c,NumOfData,50.0,90.0,90.0,100);
  { ResetAxis; }
  DrawPolygon(b,1,NumOfData,7.1,0);
  ResetAxis;
  { DrawPolygon(a,1,n,0,0,0); 
    DrawText(470,90,1,'UCL'); } 
End;

Procedure ScatterPlotGraph4;
var 
i : integer;
s : String;
Begin
  defineWindowSPC(4,54,33,78,99);
  defineHeader(4,'Raw Data'+tab5+'Window<4>'); 
  If ZoomWin[4] then redefineWindow(4,1,1,75,180);
  SelectWorld(1);
  SelectWindow(4);
  SetBackground(0);
  DrawBorder;
  DrawGraphAxis;
    for i:=1 to NumOfData do 
    begin
      c[i,1]:=i;
      c[i,2]:= SampleArray[i];
    end;
DrawPolygon(c, 1, NumOfData, -9, 2, 0);
End;

Procedure RangeGraph5;
Var i : Integer;
Begin
  defineWindowSPC(5, 1, 102, 25, 168);
  defineHeader(5, 'R-Chart' + tab5 + 'Window<5>);
  If ZoomWin[5] then RedefineWindow(5, 1, 75, 180);
  SelectWorld(1);
  SelectWindow(5);
  setBackground(0);
  DrawBorder;
  For i := 1 to NumOfData do
    begin
      c[1, 1] := i;
      c[1, 2] := SampleArray[i];
    end;
  XBar := SampleArray[50];
  UCL := SampleArray[75];
  LCL := 0;
  DrawGraphAxis;
  DrawPolygon(c, 1, NumOfData, -9, 2, 0);
End;

Procedure CuSumGraph6;
Begin
  defineWindowSPC(6, 27, 102, 51, 168);
  defineHeader(6, 'CuSum Chart' + tab5 + 'Window<6>);
  If ZoomWin[6] then RedefineWindow(6, 1, 75, 180);
  SelectWorld(3);
  SelectWindow(6);
  setBackground(0);
  DrawBorder;
  DrawGraphAxis;
  For i := 1 to NumOfData do
    begin
      c[1, 1] := i;
      c[1, 2] := SampleArray[i];
    end;
  XBar := SampleArray[50];
  UCL := SampleArray[75];
  LCL := 0;
  DrawGraphAxis;
  DrawPolygon(c, 1, NumOfData, -9, 2, 0);
DrawPolygon(b,1,n,0,0,0); }
SetLineStyle(0);
End;

Procedure Info7;
Var I:Integer;
Begin
DefineWorld(4,1,1,20,10);
defineWindowSPC(7,54,102,78,168);
defineheader(7,'Information'+tab5+'Window<7>');</
SelectWorld(4);
SelectWindow(7);
If FirstTime Then
Begin
    DrawBorder;
    setBackground(0);
    ( StartProcess:)
End
Else beep2; (Control Process:)
SetHeaderOn;
End;

Procedure PlotAllGraph:
Begin
XBarGraph2;
ParetoGraph3;
ScatterPlotGraph4;
RangeGraph5;
CuSumGraph6;
End;

procedure WindowControlAndZoom;
Label UpdateZoom;
var x, temp: real;
m, dx, dy, i, j, n, lines, scale: integer;
X1, Y1, X2, Y2: integer;
b, a: PlotArray;
Ch: Char;
begin
    ClearEol(2);
    GotoXY(12, 2);
    Write ('Use ', #24#25#26#27, ' to Move Selected Window and ESC when Done');
    CopyScreen;
    If (SelWin <> 10) And (SelWin >= MinSelWin)
    And (SelWin <= MaxSelWin) then
Begin
SelectWindow(SelWin);
Ch := ReadKey;
If Ch = #0 Then Begin
  Repeat
    Ch := ReadKey;
    Case Ch of
      #72:
        MoveVer(-5,true);
      #75:
        MoveHor(-1,true);
      #80:
        MoveVer(5,true);
      #77:
        MoveHor(1,true);
    End;
  Until Ch = #27;
End;
InvertWindow;
Ch := #0;
UpdateZoom;
SetColorWhite;
SelectSPC;
SelectWindow(10);
UpdateGraph;
ZoomWin[SelWin] := True;
Case SelWin of
  2:XBarGraph2;
  3:ParetoGraph3;
  4:ScatterPlotGraph4;
  5:RangeGraph5;
  6:CusumGraph6;
End;
ZoomWin[SelWin] := False;
SetForegroundColor(White);
GotoXY(5,1);Write ('Press RETURN to Continue ...');
Repeat
  SelectScreen(2);
  If Update Then Begin
    UpdateGraph;
    PlotAllGraph;
  End;
  SelectScreen(1);
  If Update then Goto UpdateZoom;
Until Keypressed;
Ch := ReadKey;
SetForegroundColor(7);
( SelectSPC;)
SwapScreen;
UpdateGraph;
PlotAllGraph;
End;
ClearEol(2);
end;

procedure SavePicOrTxtGrapScreen;
label 100;
var ch :Char;
   winnum.code :Integer;
procedure SaveRQPEWindow;
label 200;
var ch :String;
begin
200:
ClearEol(24);
GotoXY(5,24);
Write('Enter Window to Save:');
ch := ReadKey;
Val(ch, winnum, code);
if winnum in [minSelWin..maxSelWin] then
   begin
      ClearEol(24);
      GotoXY(5,24);
      Write('File Name To Save:');
      ch := Readkey;
      Val(ch, winnum, code);
      if ch in UserDataPath then
         begin
            ClearEol(24);
            GotoXY(50,24);
            ReadIn(ch);
            ch := Copy(ch,1,8);
            SaveWindow(WinNum,UserDataPath+ch+'.WIN');
            if idresult <> 0 then
               begin
                  Beep;
                  ClearEol(24);
                  GotoXY(5,24);
                  Write('*** Error saving file ***');
                  Goto 200;
               end;
         end;
      end;
end;
ClearEol(24);
GotoXY(5,24);
Write('Window Saved in file '+UserDataPath,ch,'.WIN',
      '...Press Return To Continue...');
ch := ReadKey;
SelectScreen(1);
end
Else
Begin
ClearEol(24);
Beep;
GotoXY(5.24);
Write ("*** Invalid Window Number ***");
Delay(25000);
End
End;

Procedure SaveROPEScreen;
Label 300:
Var ch,ch2:String;
X,Y :Integer;
Begin
300:
ClearEol(24):
GotoXY(5.24):
Write ("Enter Screen to Save 1-Displayed 2-Zoomed <1/2 >");
Ch := Readkey;
Val(CH,WinNum,Code);
ClearEol(24):
GotoXY(5.24):
Write ('File Name To Save Screen \ ',UserDataPath);
SelectWindow(9):
X := WindowX(30):
Y := WindowY(100):
GotoXY(X,Y):
ReadIn(ch):
Ch := Copy (Ch,1,8);
If WinNum = 2 then
Begin
SwapScreen;
End;
{$I-}
SaveScreen(UserDataPath+Ch+'.SCR');
{$I+}
If iOResult <> 0 then
Begin
Beep;
ClearEol(24):
GotoXY(5.24):
Write ("*** Error saving file ***");
Goto 300:
End;
If WinNum = 2 Then SwapScreen;
ClearEol(24):
GotoXY(5.24):
Write ('Window Saved in file UserDataPath,Ch+.SCR',
'...Press Return To Continue...');
Ch := ReadKey;
SelectScreen(1);
End;

begin
SelectWindow(9);
100:
ClearEol(24);
GotoXY(5,24):
Write('Save Window or Screen < W / S >');
Ch := UpCase(ReadKey);
Case Ch of
 'W' : SaveROPEWindow;
 'S' : SaveROPEScreen
Else
  Begin
    Beep;
    Goto 100;
  End;
End;
ClearEol(24);
end;

{$---}$
Procedure CheckForInput;
Var Ch : Char;
  I : Integer;
Begin
  I := 0;
  Repeat
    Inc(I);
    Until (KeyPressed) or ( I > 1000);
  If KeyPressed then Ch := ReadKey;
  If Ch = #0 Then
    Begin
      CopyScreen;
      Ch := ReadKey;
      Case Ch of
        #59:
          Begin
            SaveWindowStack('ROPEWin');
            LeaveGraphic;
            Textcolor(Yellow);
            TextBackgroundColor(Red);
            GetHelp;
            EnterGraphic;
            LoadWindowStack('ROPEWin');
            SetForegroundColor(7);
            SwapScreen;
          End;
    End;
End;
End;
#60: SelGraspWindow;
#61: WindowControlAndZoom;
#62: HardCopy(False, 0);
#63: SavePicOnTxtGrapScreen;
#64: ChangeVariables;
#68: ExitGrap := True;
Else Write (#7);
End;
End;
End;

Procedure DoROPE;
Begin
{ EdGetScreenMode;}
{ Mark(TxtPtr);
  ProcessIntro;
  Release(TxtPtr);}
InitGraphic;
NumOfData := 100;
SetClippingOn;
SetWindowModeOn;
SetHeaderOn;
SelectSPC;
ResetWindowStack;
DefineWorld(2, 0, 0, 639, 199);
SelectWorld(2);
SelWin := 10;
SelectWindow(SelWin);
  Randomize;
  XBar := SampleArray[Random(100)];
  UCL := SampleArray[Random(100)];
  LCL := SampleArray[Random(100)];
For I := MinSelWin to MaxSelWin do
  ZoomWin[I] := False;
SetUpGrapWindows;
{ UpdateGraph;}
SelectSPC;
PlotAllGraph;
Info7;
Update := False;
ExitGrap := False;
Repeat
  If Update then UpdateGraph;
  CheckForInput;
  SelectWorld(4);
  SelectWindow(7);
  Info7;
Until ExitGrap;
LeaveGraphic;
{ Close(DataFile); }
SelectWindow(9);
SetBackground(0);
DrawBorder;
SelectSPC;
GotoXY(10, 24);
  If EdYesNo ('Do you want to Shut the Process Down? ') Then
        ShutDownSystem;
End;

end. {Unit}
Unit SPC_KAL;

Interface
Uses
Dos,
Crt,
Turbo3,
Spc_ID,
Spc_Util1,
MsUser,
MsScr1n1,
Introd,
Cal;

Const CalName:Array[0..3] of String = ('Light ',
 'Panel ', 'House ', 'Storage');

Var
ATemp,VoltageOffset,Index0,Index1,Index2,Index3 : Integer;
Quit : boolean;
Therm0,Temperature0 :Array[1..MaxData] of Integer;
FFTime0 :Array[1..MaxData] of Word;
Voltage0 :Array[1..MaxData] of Real;
Therm1,Temperature1 :Array[1..MaxData] of Integer;
FFTime1 :Array[1..MaxData] of Word;
Voltage1 :Array[1..MaxData] of Real;
Therm2,Temperature2 :Array[1..MaxData] of Integer;
FFTime2 :Array[1..MaxData] of Word;
Voltage2 :Array[1..MaxData] of Real;
Therm3,Temperature3 :Array[1..MaxData] of Integer;
FFTime3 :Array[1..MaxData] of Word;
Voltage3 :Array[1..MaxData] of Real;
DataFile :Text;
TimeTakeData :Word;
UserString :String;
LineStr :Str80;
PTime :Str11;
HeapPtr :^Integer;

Procedure CalInfo;
Procedure CalScreen;
Procedure LogCalData;
Procedure CalibrateDevice;
Procedure UpdateCalibrations;
Procedure RegresXY;

{=================================================================}

Implementation

Procedure CalInfo;

Var WP:WindowRec;
    Ch:Char;

Begin
    TextColor(Yellow);
    TextBackground(Red);
    EdSaveTextWindow(Border,'How To Calibrate Sensors',5,5,75,20,WP);
    DefineTextWindow(6,6,6,69,14);
    CleanWindow(6);
    Writeln('To calibrate sensor, start up system and let run for ');  
    Writeln('about 20 minutes so that system is nearly stable. If you did');
    Writeln('not do so before now, go back to main menu and enter ROPE <F10>.');
    Writeln('If you have already prepared the system, set thermometers');
    Writeln('near sensors and be prepared to read them and enter readings');
    Writeln('at regular intervals. A suggestion is to enter temperature');
    Writeln('at markings on thermometer and avoid interpolations');
    Writeln('<< Press Return to Continue >> ');  
    Ch := ReadKey;
    EdRestoreTextWindow(WP);
End;

Procedure CalScreen;

Var I:Integer;

Begin
    DefineTextWindow(1,2,26,5);
    DefineTextWindow(2,28,2,26,5);
    DefineTextWindow(3,54,2,27,5);
    DefineTextWindow(4,20,8,40,10);
    DefineTextWindow(5,2,23,78,3);
    For I := 1 to 5 do  
        Begin
            ShowTextWindow(I);
            CleanWindow(I);
            SelWindow(I);
        End;
End;

Procedure LogCalData;

var S:Integer;

Begin
writeln(DataFile, ' ');
writeln(DataFile, ' Data collected for calibration of: ');
writeln(DataFile, CalName[0]);
writeln(DataFile, Index0);
For S := 1 to Index0 do
  writeln(DataFile, Temperature0[s]:4, Tab5, Voltage0[s]:9:6, Tab5, FFTime0[s]):
  writeln(DataFile, ' ');writeln(DataFile, ' Data collected for calibration of: ');
writeln(DataFile, CalName[1]);
writeln(DataFile, Index1);
For S := 1 to Index1 do
  writeln(DataFile, Temperature1[s]:4, Tab5, Voltage1[s]:9:6, Tab5, FFTime1[s]):
  writeln(DataFile, ' ');writeln(DataFile, ' Data collected for calibration of: ');
writeln(DataFile, CalName[2]);
writeln(DataFile, Index2);
For S := 1 to Index2 do
  writeln(DataFile, Temperature2[s]:4, Tab5, Voltage2[s]:9:6, Tab5, FFTime2[s]):
  writeln(DataFile, ' ');writeln(DataFile, ' Data collected for calibration of: ');
writeln(DataFile, CalName[3]);
writeln(DataFile, Index3);
For S := 1 to Index3 do
  writeln(DataFile, Temperature3[s]:4, Tab5, Voltage3[s]:9:6, Tab5, FFTime3[s]):
  writeln(DataFile, ' ');End;

Procedure CalibrateDevice;
Label GetChannel;

Var
  I : Integer;
  ch : Char;
  DTemp : Word;
  NoData, TakeData: Boolean;
Begin
  NoData := False;
  I := 1; Index0 := 0; Index2 := 0; Index1 := 0; Index3 := 0;
  CalScreen:
  Assign (DataFile, UserDataPath+'DataU'+UserString+'.'&&&); ($I- )
Append(DataFile);
{$1-^ >
If I0Result <> 0 Then Rewrite(DataFile);
SelWindow(5);
TextColor(14);
TextBackground(1);
PrintTime(1.1,PTime,False);
Writeln(DataFile,LineStr);
Writeln(DataFile,' ROPE Data: Iowa State University',Tab5,Tab5,
'User: ',UserData[User].Firstname,' ',UserData[User].Lastname);
Writeln(DataFile, ' ');
Writeln(DataFile,' Data collected for calibration of: ');
Writeln(DataFile,' ');
Writeln(DataFile,' 0 - Light Intensity Sensor ');
Writeln(DataFile,' 1 - Panel Temperature Sensor ');
Writeln(DataFile,' 2 - House Temperature Sensor ');
Writeln(DataFile,' 3 - Storage Temperature Sensor ');
Writeln(DataFile,LineStr);
Writeln(DataFile, ' ');
Writeln(DataFile,' Starting time = ',PTime:9);
Writeln(DataFile, ' ');
Writeln(DataFile,' DEVICE',Tab5,' TEMP ',Tab5,'VOLTAGE',Tab5,'TIME ELAPSE');
Writeln(DataFile, ' ');
InitDevice(CardNumber,Analog);
CalInfo;
Repeat
  GetTime(Hour,Minute,Second,Sec100);
  If I = 1 then
  Begin
    CalTime := Hour*Sqr(60)+Minute*60+Second;
    DTemp := CalTime;
  End
  Else
  Begin
    CalTime := Hour*Sqr(60)+Minute*60+Second;
    TimeTakeData := (CalTime - Dtemp);
  End;
  Inc(I);
  SelWindow(1);
  GotoXY(5.1);Write(CalTime:6,'s':1);
  GotoXY(5,3);
  PrintTime(5,3,PTime,True);
  Rdata := ReadAnalogData(CardNumber, Channel);
  Rdata := ConvertDataToVolts(Rdata, VoltageRange, VoltageOffset);
  SelWindow(2);
  GotoXY(2.1);
  write('Channel #',Channel:1);
  GotoXY(2.3);Write(CalName[Channel]);
  SelWindow(3);
GotoXY(1,1):Write('Voltage :'.Rdata:7:4.'VDC':3);
GotoXY(1,3):Write('Temp. last read:'.ATemp:4);
Delay(100);
GetChannel;
CleanWindown(5);
SelWindow(4);
GotoXY(1,1):Write('1  Light ');
GotoXY(1,3):Write('2 Panel ');
GotoXY(1,5):Write('3 House ');
GotoXY(1,6):Write('4 Storage');
GotoXY(1,8):Write('Enter Sensor To Calibrate --> ');
If Keypressed then
Begin
  ch := Readkey;
  Case ch of
    '1' : Channel := 0;
    '2' : Channel := 1;
    '3' : Channel := 2;
    '4' : Channel := 3;
  Else
    Begin
      SelWindow(5);
      GotoXY(2,1):Write('#7#7," *** Invalid entry ***"');
      Delay(2500);
      CleanWindow(5);
      Goto GetChannel;
    End;
End;
SelWindow(4);

Rdata := ReadAnalogData(CardNumber, Channel);
Rdata := ConvertDataToVolts( Rdata, VoltageRange, VoltageOffset );
SelWindow(5);
GotoXY(1,25);
Write ('Enter Temperature on Thermometer .. or -1 to QUIT --> ');
Readln(ATemp);
If Atemp < 0 then NoData := True
Else
  Begin
    GotoXY(1,WhereY-1);
    WriteLn(DataFile,' ',Channel,' ',ATemp:4,Tab5,Tab5,
      RData:9:6,Tab5,Tab5,TimeTakeData);
    Case Channel of
      0: begin
        Inc(Index0);
        Therm0[Index0] := Channel;
        Temperature0[Index0] := Atemp;
        Voltage0 [Index0] := RData;
        FFTime0[Index0] := TimeTakeData;
      End;
      1: begin
        Inc(Index0);
        Therm0[Index0] := Channel;
        Temperature0[Index0] := Atemp;
        Voltage0 [Index0] := RData;
        FFTime0[Index0] := TimeTakeData;
      End;
      2: begin
        Inc(Index0);
        Therm0[Index0] := Channel;
        Temperature0[Index0] := Atemp;
        Voltage0 [Index0] := RData;
        FFTime0[Index0] := TimeTakeData;
      End;
      3: begin
        Inc(Index0);
        Therm0[Index0] := Channel;
        Temperature0[Index0] := Atemp;
        Voltage0 [Index0] := RData;
        FFTime0[Index0] := TimeTakeData;
      End;
      4: begin
        Inc(Index0);
        Therm0[Index0] := Channel;
        Temperature0[Index0] := Atemp;
        Voltage0 [Index0] := RData;
        FFTime0[Index0] := TimeTakeData;
      End;
    End;
End;
1: begin
   Inc(Index1);
   Therm1[Index1] := Channel;
   Temperature1[Index1] := Atemp;
   Voltage1[Index1] := RData;
   FFTTime1[Index1] := TimeTakeData;
   End;

2: begin
   Inc(Index2);
   Therm2[Index2] := Channel;
   Temperature2[Index2] := Atemp;
   Voltage2[Index2] := RData;
   FFTTime2[Index2] := TimeTakeData;
   End;

3: begin
   Inc(Index3);
   Therm3[Index3] := Channel;
   Temperature3[Index3] := Atemp;
   Voltage3[Index3] := RData;
   FFTTime3[Index3] := TimeTakeData;
   End;

End: { Case }
SelWindow(5);
End;

Until Nodata;

LogCalData;
WriteLn(DataFile,' ');
PrintTime(1,1,PTime,False);
WriteLn(DataFile,'Ending Time : ',PTime:9);
WriteLn(DataFile,' ');
WriteLn(DataFile,LineStr);
Close(DataFile);
end; { CalibrateDevice }

Procedure UpdateCalibrations:

Var CalUpdate : Boolean;

Begin
   CleanWindow(5);
   SelWindow(5);
   GotoXY(5,1);
   Write(' Update Calibrations? [Y/N] ');
   Ch := UpCase(ReadKey);
   Repeat
      IF Ch = 'Y' then
         Begin
            InlineData := True;
         End
      Else
         Ch := UpCase(ReadKey);
   End;
End;
DataAnalysis:
End
Else
 IF Ch = 'N' then
 Begin
 End
 Else Write(#7#7):
 Until Ch in [‘Y’, ‘N’];
end;
Procedure RegresXY:
Begin ( RegresXY )
 Mark(HeapPtr):
 HelpIndex := 8;
 ATemp := 0;
 Channel := 0;
 Window(1,1,80,25):
 FillChar(LineStr,SizeOf(LineStr),#205);
 LineStr[0] := #70;
 CalibrateOevice:
 UpdateCalibrations;
 Release(HeapPtr):
End; ( RegresXY )
End. ( Unit )
Unit Cal;

Interface

Uses
  Crt.
  Dos.
  Turbo3.
  MsScrnl.
  MsUser.
  MsString.
  SPC_ID.
  Introd;

type
  CalIndex = 1..150;
  DataType = Array [CalIndex] of Real;

Var
  SPC_Cal:Text;
  SPC_File, NewFile:Text;
  CalDataX, CalDataY:DataType;
  XData, YData :Real;
  x, y, I, N, NumData,
  Instatus, OutStatus:Integer;
  RecRead :Word;
  WordRead :Word;
  ch :Char;
  z :WindowRec;
  Filename :String[64];
  InlineData :Boolean;

Procedure CalRegression;

Procedure DataAnalysis;

{==================================================================}

Implementation

Procedure CalRegression;
Label 150, 200, 300;
Var
  I :Integer;
  XY, XSqr, YSqr,
  SumX, SumY, SumXY, SumXSqr,
SumYSqr, SSqrofX,SSqrofY, XBar,YBar,Rho, BO,B1, XPoint,YPoint :Real;

Begin
  For I := 1 to N do
    Begin
      XY := CalDataX[I] * CalDataY[I];
      XSqr := SQR(CalDataX[I]);
      YSqr := SQR(CalDataY[I]);
      If I = 1 Then
        Begin
          SumX := CalDataX[I];
          SumY := CalDataY[I];
          SumXY := XY;
          SumXSqr := XSqr;
          SumYSqr := YSqr;
        End
      Else
        Begin
          SumX := SumX + CalDataX[I];
          SumY := SumY + CalDataY[I];
          SumXY := SumXY + XY;
          SumXSqr := SumXSqr + XSqr;
          SumYSqr := SumYSqr + YSqr;
        End;
    End;
  XBar := SumX/N;
  YBar := SumY/N;

  SSqrofX := (SumXSqr/N) - SQR(XBar);
  SSqrofY := (SumYSqr/N) - SQR(YBar);

  Rho := ( (SumXY/N) - XBar * YBar ) / ( SQR(SSqrofX * SSqrofY) );
  B1 := (Rho*SQRT(SSqrofY))/SQRT(SSqrofX);
  BO := YBar - B1 * XBar;

  YPoint := B1 * XPoint + BO;

  x := WhereX;
  y := WhereY;
  Gotoxy (x,y);
  ClrEol;
  Gotoxy (x+1,y);

  { Screen Dump }
150:
CleanWindow();
GotoXY(1,1);
WriteIn('X:29,Tab5,Tab5,'Y:6); 'Sum of = ':27,SumX:7:4,Tab5,SumY:7:4);
WriteIn('Sum of Squares = ':27,SumXSqr:7:4,Tab5,SumYSqr:7:4);
WriteIn('Variance of = ':27,SSqrOfX:7:4,Tab5,SSqrOfY:7:4);
WriteIn('Standard Deviation = ':27,SORT(SSqrOfX):7:4,Tab5,
SQR(SSqrOfY):7:4):
WriteIn('Mean of = ':27, XBar:7:4,Tab5,YBar:7:4);
WriteIn('Sum of XY = ':27,SumXY:7:4);
WriteIn('Correlation = ':27, Rho:7:4);
WriteIn('B0 = ':27, B0:7:4);
WriteIn('B1 = ':27, B1:7:4);
WriteIn('Regression Equation = Y = ':27,B1:7:4,'X '3,'+3,B0:7:4); CleanWindow(3);
SelWindow(3);
Write ('Do you want to Save Result ? [Y/N] ');
Ch := UpCase(ReadKey);
IF Ch = 'Y' Then Goto 200
Else
  If Ch = 'N' then Goto 300
  Else
    Begin
      Write('#7#7);
      Goto 150;
    End;
    { Print to file Newfile.Dat }
 200: 
  WriteIn(NewFile,'Sum of X = ', SumX:7:4);
  WriteIn(NewFile,'Sum of Y = ', SumY:7:4);
  WriteIn(NewFile,'Sum of XY = ', SumXY:7:4);
  WriteIn(NewFile,'Sum of X Squared = ',SumXSqr:7:4);
  WriteIn(NewFile,'Sum of Y Squared = ',SumYSqr:7:4);
  WriteIn(NewFile,').
  WriteIn(NewFile,'Variance of X = ',SSqrOfX:7:4);
  WriteIn(NewFile,'Variance of Y = ',SSqrOfY:7:4);
  WriteIn(NewFile,').
  WriteIn(NewFile,'Standard Deviation of X = ',SQR(SSqrOfX):7:4);
  WriteIn(NewFile,'Standard Deviation of Y = ',SQR(SSqrOfY):7:4);
  WriteIn(NewFile,').
  WriteIn(NewFile,'Mean of X = ', XBar:7:4);
  WriteIn(NewFile,'Mean of Y = ', YBar:7:4);
  WriteIn(NewFile,'Correlation = ', Rho:7:4);
  WriteIn(NewFile,'BO = ', B0:7:4);
  WriteIn(NewFile,'B1 = ', B1:7:4);
  WriteIn(NewFile,').
  WriteIn(NewFile,'Regression Equation => Y = ':5,B1:7:4,'X '3,' +3,B0:7:4);
WriteIn(NewFile);
300:
end;

Procedure DataAnalysis;

Label 100:

Begin
100:
TextColor(Yellow);
TextBackground(Blue);
DefineTextWindow (1.3,2,60.17);
DefineTextWindow (2.20.5,70.18);
DefineTextWindow (3.3,20.78.5);
EdSaveTextWindow (Border,'Data Analysis ',1.1,80.25.z);
EdSetCursor($0607):
If Not InlineData then
Begin
  GotoXY(3,12):ClrEol;
  Write ('Enter the Name of your Data File ...');
  GotoXY(40,12):ClrEol;
  Readln(Filename);
End
Else
  FileName := UserDataPath+'DataU'+UserString+'.dat';
CleanWindow(3);
SelWindow(3);
GotoXY(1,4):Write ('Input Data File ...',Filename);
GotoXY(1,5):Write ('Output Data File ...',UserDataPath+'StatU'+UserString+'.Dat');
($1-$)
Assign (SPC_File,FileName);
Assign (NewFile,UserDataPath+'StatU'+UserString+'.dat');
Reset (SPC_File):
InStatus := IOResult;
If FirstTime then Rewrite (NewFile)
Else Append (NewFile);
OutStatus := IOResult;
($1+$)
N := 1;
x := WhereX;
y := WhereY;
GotoXY (1,1);
ClrEol;
If InStatus <> 0 then
Begin
  TextBackground(Red);
  CleanWindow(3):
SelWindow(3);
WriteIn (#7#7.' *** Input File Not Found / Invalid file Name ***');
Write (#7#7.' Press any key to Continue ... ');
TextBackground(blue);
Ch := Readkey;
Goto 100;
End;
If OutStatus <> 0 then
Begin
TextBackGround(Red);
CleanWindow(3);
SelWindow(3);
Write (#7#7.' *** File Creation Error *** ');
TextBackground(blue);
Delay(5000);
Exit;
End;
Write (#7.' Working Please Wait ... ');
WriteIn (NewFile);
WriteIn (NewFile,' DATA ');
WriteIn (NewFile);
WriteIn (NewFile,' X Y ');
WriteIn (NewFile);
GotoXY (1,1);
Read (SPC_File,NumData);
ReadIn (SPC_File);
CleanWindow(1);
SelWindow(1);
WriteIn;
WriteIn (#7.' DATA ');
WriteIn;
WriteIn (#7.' X Y ');
WriteIn;
While (N <= NumData) do
Begin
Read (SPC_File,CalDataY[N]);
Read (SPC_File,CalDataX[N]);
ReadIn (SPC_File);
WriteIn (CalDataX[N]:7:4, ' ', CalDataY[N]:7:4);
WriteIn (NewFile,CalDataX[N]:7:4, ' ', CalDataY[N]:7:4);
N := N + 1;
Delay(100);
End;
N := N - 1;
WriteIn (NewFile);
WriteIn (NewFile,---------- END DATA ---------- ');
WriteIn (NewFile);
Close (SPC_File);
CalRegression;
Repeat
CleanWindow(3);
SelWindow(3);
Write(’Do you want to do another calculation? [Y/N]’);
Ch := UpCase(ReadKey);
If Ch = ’Y’ then Goto 100
Else If Ch = ’N’ then Write(’’)
    Else Write(#7#7);
    Until Ch in [’Y’,’N’];
    Close(NewFile);
    EdRestoreTextWindow(z);
End;

End. { Cal }
APPENDIX D. DATA FROM THE CALIBRATION OF THE SENSORS
## Data for House Thermistor Calibration

<table>
<thead>
<tr>
<th>TEMP.</th>
<th>V</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3.515625</td>
<td>12.35962</td>
<td>43.45179</td>
<td>152.7602</td>
<td>50.58937</td>
</tr>
<tr>
<td>64</td>
<td>3.256836</td>
<td>10.60698</td>
<td>34.5452</td>
<td>112.508</td>
<td>63.12594</td>
</tr>
<tr>
<td>78</td>
<td>2.84668</td>
<td>8.103587</td>
<td>23.06832</td>
<td>65.66812</td>
<td>77.7259</td>
</tr>
<tr>
<td>80</td>
<td>2.77832</td>
<td>7.719062</td>
<td>21.44602</td>
<td>59.58392</td>
<td>79.81761</td>
</tr>
<tr>
<td>82</td>
<td>2.719727</td>
<td>7.396915</td>
<td>20.11759</td>
<td>54.71435</td>
<td>81.57461</td>
</tr>
<tr>
<td>84</td>
<td>2.636719</td>
<td>6.952287</td>
<td>18.33123</td>
<td>48.3343</td>
<td>84.03015</td>
</tr>
<tr>
<td>86</td>
<td>2.563477</td>
<td>6.571414</td>
<td>16.84567</td>
<td>43.18349</td>
<td>86.18591</td>
</tr>
<tr>
<td>88</td>
<td>2.509766</td>
<td>6.298925</td>
<td>15.80883</td>
<td>39.67646</td>
<td>87.77231</td>
</tr>
<tr>
<td>90</td>
<td>2.446289</td>
<td>5.984333</td>
<td>14.6394</td>
<td>35.8122</td>
<td>89.66436</td>
</tr>
<tr>
<td>92</td>
<td>2.353516</td>
<td>5.539038</td>
<td>13.03621</td>
<td>30.68094</td>
<td>92.48566</td>
</tr>
<tr>
<td>94</td>
<td>2.280273</td>
<td>5.199645</td>
<td>11.85661</td>
<td>27.03631</td>
<td>94.77925</td>
</tr>
<tr>
<td>96</td>
<td>2.22168</td>
<td>4.935862</td>
<td>10.96591</td>
<td>24.36273</td>
<td>96.66721</td>
</tr>
<tr>
<td>100</td>
<td>2.089844</td>
<td>4.367448</td>
<td>9.127285</td>
<td>19.0746</td>
<td>101.1259</td>
</tr>
<tr>
<td>102</td>
<td>2.001953</td>
<td>4.007816</td>
<td>8.023459</td>
<td>16.06259</td>
<td>104.2879</td>
</tr>
<tr>
<td>104</td>
<td>1.933594</td>
<td>3.738786</td>
<td>7.229294</td>
<td>13.97852</td>
<td>106.866</td>
</tr>
<tr>
<td>106</td>
<td>1.870117</td>
<td>3.497338</td>
<td>6.54043</td>
<td>12.23137</td>
<td>109.3601</td>
</tr>
<tr>
<td>110</td>
<td>1.811523</td>
<td>3.281618</td>
<td>5.944722</td>
<td>10.769</td>
<td>111.7525</td>
</tr>
<tr>
<td>112</td>
<td>1.831055</td>
<td>3.352762</td>
<td>6.139092</td>
<td>11.24102</td>
<td>110.9451</td>
</tr>
<tr>
<td>114</td>
<td>1.796875</td>
<td>3.22876</td>
<td>5.801678</td>
<td>10.42489</td>
<td>112.3646</td>
</tr>
<tr>
<td>116</td>
<td>1.743164</td>
<td>3.038621</td>
<td>5.296814</td>
<td>9.23216</td>
<td>114.6579</td>
</tr>
<tr>
<td>118</td>
<td>1.767578</td>
<td>3.124332</td>
<td>5.5225</td>
<td>9.76145</td>
<td>113.6058</td>
</tr>
<tr>
<td>120</td>
<td>1.708984</td>
<td>2.920626</td>
<td>4.991304</td>
<td>8.530058</td>
<td>116.1582</td>
</tr>
<tr>
<td>122</td>
<td>1.68457</td>
<td>2.837776</td>
<td>4.780432</td>
<td>8.052793</td>
<td>117.2496</td>
</tr>
<tr>
<td>124</td>
<td>1.665039</td>
<td>2.772355</td>
<td>4.616079</td>
<td>7.685952</td>
<td>118.1348</td>
</tr>
<tr>
<td>126</td>
<td>1.645508</td>
<td>2.707697</td>
<td>4.455536</td>
<td>7.331621</td>
<td>119.0307</td>
</tr>
<tr>
<td>128</td>
<td>1.606445</td>
<td>2.580666</td>
<td>4.145697</td>
<td>6.659835</td>
<td>120.8552</td>
</tr>
<tr>
<td>130</td>
<td>1.591797</td>
<td>2.533818</td>
<td>4.033323</td>
<td>6.420232</td>
<td>121.5507</td>
</tr>
<tr>
<td>132</td>
<td>1.542969</td>
<td>2.380753</td>
<td>3.673429</td>
<td>5.667986</td>
<td>123.9137</td>
</tr>
<tr>
<td>134</td>
<td>1.499023</td>
<td>2.240707</td>
<td>3.36841</td>
<td>5.049323</td>
<td>126.0998</td>
</tr>
<tr>
<td>136</td>
<td>1.489258</td>
<td>2.217889</td>
<td>3.30301</td>
<td>4.919033</td>
<td>126.5932</td>
</tr>
<tr>
<td>138</td>
<td>1.450195</td>
<td>2.103066</td>
<td>3.049855</td>
<td>4.422885</td>
<td>128.5952</td>
</tr>
<tr>
<td>140</td>
<td>1.416016</td>
<td>2.005101</td>
<td>2.839256</td>
<td>4.020431</td>
<td>130.3836</td>
</tr>
<tr>
<td>142</td>
<td>1.362305</td>
<td>1.855875</td>
<td>2.528268</td>
<td>3.444272</td>
<td>133.2635</td>
</tr>
<tr>
<td>144</td>
<td>1.308594</td>
<td>1.712418</td>
<td>2.24086</td>
<td>2.932376</td>
<td>136.2279</td>
</tr>
<tr>
<td>146</td>
<td>1.264648</td>
<td>1.599335</td>
<td>2.022595</td>
<td>2.557871</td>
<td>138.7159</td>
</tr>
<tr>
<td>148</td>
<td>1.230469</td>
<td>1.514054</td>
<td>1.862996</td>
<td>2.292359</td>
<td>140.6894</td>
</tr>
<tr>
<td>150</td>
<td>1.206055</td>
<td>1.454569</td>
<td>1.75429</td>
<td>2.11577</td>
<td>142.1196</td>
</tr>
<tr>
<td>152</td>
<td>1.196289</td>
<td>1.431107</td>
<td>1.712018</td>
<td>2.048068</td>
<td>142.9664</td>
</tr>
<tr>
<td>154</td>
<td>1.176758</td>
<td>1.384759</td>
<td>1.629527</td>
<td>1.917559</td>
<td>143.8581</td>
</tr>
<tr>
<td>156</td>
<td>1.157227</td>
<td>1.339174</td>
<td>1.549729</td>
<td>1.793388</td>
<td>145.0305</td>
</tr>
<tr>
<td>158</td>
<td>1.132813</td>
<td>1.283265</td>
<td>1.4537</td>
<td>1.64677</td>
<td>146.5109</td>
</tr>
<tr>
<td>160</td>
<td>1.103516</td>
<td>1.217748</td>
<td>1.343804</td>
<td>1.482909</td>
<td>148.3091</td>
</tr>
<tr>
<td>162</td>
<td>1.083984</td>
<td>1.175021</td>
<td>1.273704</td>
<td>1.380675</td>
<td>149.5208</td>
</tr>
</tbody>
</table>

Regression Output:
Data for Panel Thermistor Calibration

<table>
<thead>
<tr>
<th>TEMP.</th>
<th>V</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>3.4082</td>
<td>11.6158</td>
<td>39.5892</td>
<td>134.9279</td>
<td>47.8789</td>
</tr>
<tr>
<td>48</td>
<td>3.3936</td>
<td>11.5162</td>
<td>39.0809</td>
<td>132.6232</td>
<td>48.6192</td>
</tr>
<tr>
<td>54</td>
<td>3.2764</td>
<td>10.7346</td>
<td>35.1704</td>
<td>115.2312</td>
<td>54.4096</td>
</tr>
<tr>
<td>56</td>
<td>3.2520</td>
<td>10.5752</td>
<td>34.3900</td>
<td>111.8348</td>
<td>55.5886</td>
</tr>
<tr>
<td>58</td>
<td>3.2227</td>
<td>10.3855</td>
<td>33.4689</td>
<td>107.8589</td>
<td>56.9922</td>
</tr>
<tr>
<td>60</td>
<td>3.1689</td>
<td>10.0422</td>
<td>31.8232</td>
<td>100.8460</td>
<td>53.5350</td>
</tr>
<tr>
<td>62</td>
<td>3.1152</td>
<td>9.7047</td>
<td>30.2324</td>
<td>94.1809</td>
<td>62.0412</td>
</tr>
<tr>
<td>64</td>
<td>3.0713</td>
<td>9.4328</td>
<td>28.9709</td>
<td>88.9780</td>
<td>64.0665</td>
</tr>
<tr>
<td>66</td>
<td>3.0029</td>
<td>9.0176</td>
<td>27.0792</td>
<td>81.3169</td>
<td>67.1757</td>
</tr>
<tr>
<td>66</td>
<td>3.0273</td>
<td>9.1648</td>
<td>27.7474</td>
<td>83.9938</td>
<td>66.0708</td>
</tr>
<tr>
<td>70</td>
<td>2.9688</td>
<td>8.8135</td>
<td>26.1650</td>
<td>77.6774</td>
<td>68.7126</td>
</tr>
<tr>
<td>72</td>
<td>2.9150</td>
<td>8.4975</td>
<td>24.7704</td>
<td>72.2067</td>
<td>71.1059</td>
</tr>
<tr>
<td>74</td>
<td>2.8467</td>
<td>8.1036</td>
<td>23.0683</td>
<td>65.6681</td>
<td>74.1154</td>
</tr>
<tr>
<td>76</td>
<td>2.7734</td>
<td>7.6920</td>
<td>21.3332</td>
<td>59.1662</td>
<td>77.2978</td>
</tr>
<tr>
<td>76</td>
<td>2.7686</td>
<td>7.6649</td>
<td>21.2207</td>
<td>58.7506</td>
<td>77.5085</td>
</tr>
<tr>
<td>80</td>
<td>2.6904</td>
<td>7.2384</td>
<td>19.4744</td>
<td>52.3946</td>
<td>80.8555</td>
</tr>
<tr>
<td>82</td>
<td>2.6465</td>
<td>7.0039</td>
<td>18.5356</td>
<td>49.0543</td>
<td>82.7188</td>
</tr>
<tr>
<td>84</td>
<td>2.6123</td>
<td>6.8241</td>
<td>17.8267</td>
<td>46.5689</td>
<td>84.1585</td>
</tr>
<tr>
<td>86</td>
<td>2.5977</td>
<td>6.7478</td>
<td>17.5285</td>
<td>45.5330</td>
<td>84.7730</td>
</tr>
<tr>
<td>88</td>
<td>2.5537</td>
<td>6.5214</td>
<td>16.6539</td>
<td>42.5292</td>
<td>86.6075</td>
</tr>
<tr>
<td>90</td>
<td>2.5049</td>
<td>6.2744</td>
<td>15.7167</td>
<td>39.3686</td>
<td>88.6297</td>
</tr>
<tr>
<td>92</td>
<td>2.4463</td>
<td>5.9843</td>
<td>14.6394</td>
<td>35.8122</td>
<td>91.0336</td>
</tr>
<tr>
<td>94</td>
<td>2.3828</td>
<td>5.6778</td>
<td>13.5291</td>
<td>32.2374</td>
<td>93.6088</td>
</tr>
<tr>
<td>96</td>
<td>2.3242</td>
<td>5.4020</td>
<td>12.5554</td>
<td>29.1815</td>
<td>95.9579</td>
</tr>
<tr>
<td>98</td>
<td>2.2461</td>
<td>5.0449</td>
<td>11.3314</td>
<td>25.4514</td>
<td>99.0448</td>
</tr>
<tr>
<td>100</td>
<td>2.1875</td>
<td>4.7852</td>
<td>10.4675</td>
<td>22.8977</td>
<td>101.3233</td>
</tr>
<tr>
<td>100</td>
<td>2.2021</td>
<td>4.8495</td>
<td>10.6792</td>
<td>23.5172</td>
<td>100.7568</td>
</tr>
<tr>
<td>102</td>
<td>2.1582</td>
<td>4.6578</td>
<td>10.0526</td>
<td>21.6955</td>
<td>102.4498</td>
</tr>
<tr>
<td>104</td>
<td>2.1094</td>
<td>4.4495</td>
<td>9.3856</td>
<td>19.7977</td>
<td>104.3072</td>
</tr>
<tr>
<td>106</td>
<td>2.0801</td>
<td>4.3267</td>
<td>8.9999</td>
<td>18.7205</td>
<td>105.4089</td>
</tr>
<tr>
<td>108</td>
<td>2.0508</td>
<td>4.2057</td>
<td>8.6250</td>
<td>17.6879</td>
<td>106.5005</td>
</tr>
<tr>
<td>110</td>
<td>1.9482</td>
<td>3.7956</td>
<td>7.3948</td>
<td>14.4069</td>
<td>110.2346</td>
</tr>
</tbody>
</table>

Regression Output:

| Constant | 227.2725 |
| Std Err of Y Est | 1.429579 |
| R Squared | 0.996819 |
| No. of Observations | 44 |
| Degrees of Freedom | 39 |
| X Coefficient(s) | -73.7381 |
| Std Err of Coef. | 55.53498 |
| Data for Panel Thermistor Calibration | 40.75976 |
| R Squared | 12.6845 |

Regression Output:

| Constant | 103.5134 |
| Std Err of Y Est | 0.9278 |
| R Squared | 0.9978 |
| No. of Observations | 33.0000 |
| Degrees of Freedom | 28.0000 |
### Data for Storage Thermistor Calibration

<table>
<thead>
<tr>
<th>TEMP.</th>
<th>V</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>PREDICTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>3.7207</td>
<td>13.8436</td>
<td>51.5080</td>
<td>191.6461</td>
<td>48.4924</td>
</tr>
<tr>
<td>48</td>
<td>3.7256</td>
<td>13.8800</td>
<td>51.7111</td>
<td>192.6542</td>
<td>48.1797</td>
</tr>
<tr>
<td>50</td>
<td>3.7061</td>
<td>13.7348</td>
<td>50.9021</td>
<td>188.6459</td>
<td>49.4101</td>
</tr>
<tr>
<td>52</td>
<td>3.6768</td>
<td>13.5185</td>
<td>49.7044</td>
<td>182.7512</td>
<td>51.1557</td>
</tr>
<tr>
<td>54</td>
<td>3.6279</td>
<td>13.1619</td>
<td>47.7504</td>
<td>173.2350</td>
<td>53.8147</td>
</tr>
<tr>
<td>56</td>
<td>3.5840</td>
<td>12.8449</td>
<td>46.0361</td>
<td>164.9925</td>
<td>55.9615</td>
</tr>
<tr>
<td>58</td>
<td>3.5205</td>
<td>12.3940</td>
<td>43.6331</td>
<td>153.6107</td>
<td>58.6944</td>
</tr>
<tr>
<td>60</td>
<td>3.4570</td>
<td>11.9511</td>
<td>41.3152</td>
<td>142.8279</td>
<td>61.0466</td>
</tr>
<tr>
<td>64</td>
<td>3.3398</td>
<td>11.1546</td>
<td>37.2545</td>
<td>124.4242</td>
<td>64.5800</td>
</tr>
<tr>
<td>66</td>
<td>3.3008</td>
<td>10.8952</td>
<td>35.9625</td>
<td>118.7044</td>
<td>65.5711</td>
</tr>
<tr>
<td>68</td>
<td>3.2373</td>
<td>10.4801</td>
<td>33.9274</td>
<td>109.8334</td>
<td>67.0287</td>
</tr>
<tr>
<td>70</td>
<td>3.1592</td>
<td>9.9804</td>
<td>31.5299</td>
<td>99.6087</td>
<td>68.6225</td>
</tr>
<tr>
<td>72</td>
<td>3.1242</td>
<td>9.4020</td>
<td>29.5554</td>
<td>92.1815</td>
<td>67.8473</td>
</tr>
<tr>
<td>74</td>
<td>3.0949</td>
<td>8.6267</td>
<td>27.0866</td>
<td>84.0988</td>
<td>68.9088</td>
</tr>
<tr>
<td>76</td>
<td>3.0651</td>
<td>7.9513</td>
<td>24.3172</td>
<td>76.5121</td>
<td>69.9331</td>
</tr>
<tr>
<td>78</td>
<td>3.0358</td>
<td>7.2769</td>
<td>21.5478</td>
<td>69.6210</td>
<td>71.6935</td>
</tr>
<tr>
<td>80</td>
<td>3.0065</td>
<td>6.6035</td>
<td>18.7884</td>
<td>63.3800</td>
<td>73.3639</td>
</tr>
<tr>
<td>82</td>
<td>2.9772</td>
<td>5.9311</td>
<td>16.0390</td>
<td>57.7400</td>
<td>75.0242</td>
</tr>
<tr>
<td>84</td>
<td>2.9479</td>
<td>5.2597</td>
<td>13.3006</td>
<td>52.7100</td>
<td>76.6843</td>
</tr>
<tr>
<td>86</td>
<td>2.9186</td>
<td>4.5893</td>
<td>10.5722</td>
<td>48.3100</td>
<td>78.3447</td>
</tr>
<tr>
<td>88</td>
<td>2.8893</td>
<td>3.9199</td>
<td>7.8538</td>
<td>44.5400</td>
<td>80.0051</td>
</tr>
<tr>
<td>90</td>
<td>2.8600</td>
<td>3.2515</td>
<td>5.1454</td>
<td>41.4100</td>
<td>81.6657</td>
</tr>
<tr>
<td>92</td>
<td>2.8307</td>
<td>2.5841</td>
<td>2.4370</td>
<td>38.9000</td>
<td>83.3262</td>
</tr>
<tr>
<td>94</td>
<td>2.8014</td>
<td>1.9167</td>
<td>0.7286</td>
<td>36.9900</td>
<td>84.9870</td>
</tr>
<tr>
<td>96</td>
<td>2.7721</td>
<td>1.2493</td>
<td>-0.2814</td>
<td>35.7700</td>
<td>86.6479</td>
</tr>
<tr>
<td>98</td>
<td>2.7428</td>
<td>0.5821</td>
<td>-0.9528</td>
<td>34.3600</td>
<td>88.3088</td>
</tr>
<tr>
<td>100</td>
<td>2.7135</td>
<td>-0.2467</td>
<td>-2.6242</td>
<td>33.7500</td>
<td>90.0698</td>
</tr>
<tr>
<td>102</td>
<td>2.6842</td>
<td>-0.9343</td>
<td>-5.2956</td>
<td>33.0400</td>
<td>91.8309</td>
</tr>
<tr>
<td>104</td>
<td>2.6549</td>
<td>-1.6219</td>
<td>-7.9670</td>
<td>32.3300</td>
<td>93.6020</td>
</tr>
<tr>
<td>106</td>
<td>2.6256</td>
<td>-2.3095</td>
<td>-10.6384</td>
<td>31.6200</td>
<td>95.3732</td>
</tr>
<tr>
<td>108</td>
<td>2.5963</td>
<td>-2.9971</td>
<td>-13.3098</td>
<td>30.9100</td>
<td>97.1443</td>
</tr>
<tr>
<td>112</td>
<td>2.5377</td>
<td>-4.3723</td>
<td>-18.6526</td>
<td>29.4900</td>
<td>100.6867</td>
</tr>
<tr>
<td>114</td>
<td>2.5084</td>
<td>-5.0600</td>
<td>-21.3240</td>
<td>28.7800</td>
<td>102.4580</td>
</tr>
<tr>
<td>116</td>
<td>2.4791</td>
<td>-5.7476</td>
<td>-23.9954</td>
<td>28.0700</td>
<td>104.2293</td>
</tr>
<tr>
<td>118</td>
<td>2.4498</td>
<td>-6.4352</td>
<td>-26.6668</td>
<td>27.3600</td>
<td>106.0006</td>
</tr>
<tr>
<td>120</td>
<td>2.4205</td>
<td>-7.1228</td>
<td>-29.3382</td>
<td>26.6500</td>
<td>107.7720</td>
</tr>
<tr>
<td>122</td>
<td>2.3912</td>
<td>-7.8104</td>
<td>-32.0096</td>
<td>25.9400</td>
<td>109.5434</td>
</tr>
<tr>
<td>124</td>
<td>2.3619</td>
<td>-8.4980</td>
<td>-34.6810</td>
<td>25.2300</td>
<td>111.3148</td>
</tr>
<tr>
<td>126</td>
<td>2.3326</td>
<td>-9.1856</td>
<td>-37.3524</td>
<td>24.5200</td>
<td>113.0862</td>
</tr>
<tr>
<td>130</td>
<td>2.2740</td>
<td>-10.5608</td>
<td>-42.5152</td>
<td>23.1000</td>
<td>116.6290</td>
</tr>
<tr>
<td>132</td>
<td>2.2447</td>
<td>-11.2484</td>
<td>-45.0966</td>
<td>22.3900</td>
<td>118.3904</td>
</tr>
<tr>
<td>134</td>
<td>2.2154</td>
<td>-11.9360</td>
<td>-47.6780</td>
<td>21.6800</td>
<td>120.1619</td>
</tr>
</tbody>
</table>

### Regression Output:

- **Constant**: 24.7719
- **Std Err of Y Est**: 0.7445
- **R Squared**: 0.9994
- **No. of Observations**: 35.0000
- **Degrees of Freedom**: 30.0000
- **X Coefficient(s)**: 302.9336 -254.93 77.8309
| Std Err of Coef. | 50.0917 | 31.4196 | 8.4016 |
APPENDIX E. ROPE USER MANUAL
Introduction
Welcome to ROPE. With this system, you will be able to use SPC to control a process. You will learn how setting various tolerance limits will affect the running of a process. In this case, the process is keeping the temperature of a house at 85°C. The heating is done using solar energy (light); excess heat may be stored for later use.

Use the main power switch to turn on the computer system. Enter the command <ROPE><cr> at the Dos prompt. After a short wait, the ROPE logo will appear (See Figure 31). You will then be prompted for the logon information- username and password (See Figure 32). At this level you may change your password, and perhaps perform other functions depending on your status and assign mode. Table E-1 and E-2 show which functions are available to which users.

Function table

Table E-1. Function Available at the Logon Level.

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>FUNCTION</th>
<th>KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Make User List</td>
<td>&lt;F1&gt;</td>
</tr>
<tr>
<td>2</td>
<td>Add Users</td>
<td>&lt;F3&gt;&lt;F6&gt;</td>
</tr>
<tr>
<td>3</td>
<td>Delete Users</td>
<td>&lt;F3&gt;&lt;F8&gt;</td>
</tr>
<tr>
<td>4</td>
<td>Change User Information</td>
<td>&lt;F3&gt;&lt;F5&gt;</td>
</tr>
<tr>
<td>5</td>
<td>Change Password</td>
<td>&lt;F7&gt;</td>
</tr>
<tr>
<td>6</td>
<td>List Users</td>
<td>&lt;F2&gt;</td>
</tr>
<tr>
<td>7</td>
<td>Enable Quiting of Program</td>
<td>&lt;F3&gt;&lt;F10&gt;</td>
</tr>
<tr>
<td>8</td>
<td>Continue</td>
<td>&lt;F10&gt;</td>
</tr>
</tbody>
</table>

Table E-2. Functions Available to Users of different Modes and Status.

<table>
<thead>
<tr>
<th>MODE</th>
<th>PROGRAMMER</th>
<th>PROFESSOR</th>
<th>GRADUATE</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected</td>
<td>N/A</td>
<td>N/A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Available</td>
<td>A</td>
<td>A</td>
<td>5,8</td>
<td>5,8</td>
</tr>
<tr>
<td>Unprotected</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td>1-8</td>
<td>1-8</td>
<td>1,5,6,8</td>
<td>5,6,8</td>
</tr>
<tr>
<td>Available</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After passing through this level, the introduction is displayed. The main menu then appears (see Figure 33). At this point, you should have a data Disk in the A drive.

Help <F1>
Pressing the F1 key at any time will give you help on the section of the program that you are working. You can page through all of the help file by using the arrow keys:

- **Home**: will take you to the beginning of the help file.
- **End**: will take you to the end of the help file.
- **PgUp**: moves up the screen and display a screen of help text.
- **PgDn**: moves down the screen and display a screen of help text.
- **UpArrow**: moves up a line.
- **DownArrow**: moves down a line.

**Statistic <F2>**

This section allows you to do Bar graphs, linear regression, pie charts and linear graphs of data from any datafile or from the keyboard. The functions and keys are shown in Table E-3. You will be prompted for the name of the file where your data is stored.

**Table E-3. Statistics Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pie Chart</td>
<td>&lt;F2&gt;</td>
</tr>
<tr>
<td>Linear Graph (XY)</td>
<td>&lt;F3&gt;</td>
</tr>
<tr>
<td>Bar Graph</td>
<td>&lt;F4&gt;</td>
</tr>
<tr>
<td>Analysis(X-Bar, e, r)</td>
<td>&lt;F5&gt;</td>
</tr>
<tr>
<td>From DataFile</td>
<td>&lt;F6&gt;</td>
</tr>
<tr>
<td>From Keyboard</td>
<td>&lt;F7&gt;</td>
</tr>
<tr>
<td>Save Results</td>
<td>&lt;F8&gt;</td>
</tr>
</tbody>
</table>

**Calibration <F3>**

This segment allows for the simultaneous calibration of the sensors used to detect temperature of the house, solar panel, storage liquid, and light intensity of the solar panel. Inputting temperatures read from a thermometer makes the computer record a corresponding voltage. The voltages are then correlated with the temperatures. In the actual SPC process, the computer uses the regression equation calculated from these readings. This segment is menu driven and therefore needs very little explanation (See Figure 34).

**Editor <F4>**

This editor is here incase you have to edit any files - for example - files made by the calibration and statistic segments. It may also be used to print text files. F10 activates the main menu of the editor; further instructions may be found in the Turbo Pascal Editor ToolBox by Borland International. The editor has an on-line help under the main menu option MicroStar.
Shell to DOS <F5>

After pressing <F5>, you may perform any DOS commands. There are 2 levels of shelling as follows:

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>TO DOS</th>
<th>FROM DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;F5&gt; (from ROPE)</td>
<td>&lt;ROPE&gt; (to ROPE)</td>
</tr>
<tr>
<td>2</td>
<td>&lt;cr&gt; (from Level 1)</td>
<td>&lt;EXIT&gt; (to Level 1)</td>
</tr>
<tr>
<td></td>
<td>or &lt;F5&gt;&lt;cr&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Exit <F6>

This function will let you exit ROPE, however, you can only do so by first enabling the quit mode. To enable the quit mode you must be of the status of professor or in the unprotected mode. By entering <F3>-<F10> in the introduction you can activate this function.

If you are not in either of the above mode this function will clean up the files, prompt the user to shut the system down, and reset the program for the next user.
APPENDIX  F. INFORMATION USED TO PROGRAM THE DACA BOARD
IBM PC Data Acquisition and Control Adapter and Software

Introduction

Personal computers have become common tools in business and industry over the past few years. These computers have begun to be integrated into workstations to allow more creative and productive use of them in the laboratories of scientists and engineers. In order to aid these users, IBM has developed hardware and software products resulting in the IBM PC Engineering/Scientific Series. These products allow the user to customize a PC into a personal, versatile, productive tool that can be used in both the laboratory and the office.

Two Hardware Products

Two products allow direct connection of IBM Personal Computers to instrumentation, sensors and control signals. The IBM PC Data Acquisition and Control (DAC) Adapter provides the capability to control and monitor processes within a sensor based system. Also available as an option for use with the DAC Adapter is a DAC Adapter Distribution Panel.

The General Purpose Interface Bus (GPIB) Adapter which allows access to and control of instruments and devices that are compatible with the IEEE-488 standard.

Two Software Products

A programming support product is also provided for the DAC and GPIB Adapters to enable efficient application development.

The IBM PC Data Acquisition and Control Adapter Programming Support provides an easy to use interface for accessing the analog and digital I/O onboard and expansion capabilities of the DAC Adapter.

The IBM PC General Purpose Interface Bus Adapter Programming Support provides appropriate levels of capability for both primitive functions that handle GPIB activities and high level functions that contribute to ease of use. The result can be a highly efficient workstation that can help improve accuracy and productivity in even complex testing or measurement systems using the GPIB Adapter.

The IBM PC DAC and GPIB Adapters and their programming support products enable engineers, scientists and technicians to use personal computers from IBM in both process and instrument control. The capabilities of the adapters and the real time software coupled with the open architecture and versatility of the line of IBM Personal Computers, offer new opportunities for designing workstations that can support data acquisition, control, analysis and quality control testing activities in the lab, the pilot plant or the full-scale production line. In addition to the ease of use of obtaining data and automating the laboratory through the IBM PC DAC and GPIB Adapters, the user can utilize the power of the IBM PC family for data analysis and data base management.

The remainder of this article will be dedicated to a description of the IBM PC Data Acquisition and Control Adapter and its software support. The next article will describe more fully the IBM PC General Purpose Interface Bus Adapter and its software support.

IBM Data Acquisition and Control Adapter

The engineering/scientific workstation can use the IBM PC DAC Adapter to interface the digital computer with analog laboratory testing and process control equipment. Using a single adapter and appropriate software enables the IBM PC to monitor and control analog and discrete signals. Four DAC Adapters can be used with a single IBM Personal Computer.

Data is acquired and processes controlled by the DAC Adapter which integrates analog, binary and timer/counter devices on a single adapter card, which plugs into any full length slot in an IBM PC AT, IBM PC XT, IBM PC or IBM Portable PC. Generally, the information being read by the computer is analog in nature. In order for the computer to be able to process this information it must first be translated into digital data. This is the function of the analog-to-digital (A/D) converter. The digital-to-analog (D/A) converter and binary I/O devices allow the user to control the operation of any process.
Analog-To-Digital and Digital-To-Analog Conversion

An analog-to-digital conversion subsystem or analog input will convert voltages over a given range to digital values. The three selectable ranges allowed are:

-5 to +5 volts / -10 to +10 volts / 0 to +10 volts

The digital-to-analog subsystem or analog output converts digital values to the selectable values referenced above.

Binary Input/Output

A binary I/O device consists of two subsystems, one for input and one for output. The device has a 16-bit input port, a 16-bit output port, I/O handshaking capabilities and optional external clocking/trigging. This device will sense the state of discrete signals and control devices that require control signals.

Timer/Counter Device

The 8253-5 timer/counter device which can be programmed includes three 16-bit timers/counters: 0, 1 and 2. These timer/counters are used to provide pulses, count events when pulsed and, through software, generate interrupts to the DAC Distribution Panel. Counters 0 and 1 are cascaded together to perform internal clocking functions. Counter 2 is used as a countdown timer.

Additional Features

Additional user features of the DAC Adapter include:
- A system reference voltage.
- Optional screw terminal DAC Distribution Panel for easy access to I/O signals.
- Diagnostic wrap connector for easy installation verification and ongoing reverification of I/O signal levels.
- Up to four IBM DAC Adapters in a system.

The DAC Adapter features an expansion bus interface that consists of two 34-pin transition connectors and provides a full 16-bit parallel data path and addressing for up to 253 expansion devices.

IBM PC DAC Adapter Interface Specifications

The Data Acquisition and Control Adapter utilizes IBM PC I/O channels as described in the Technical Reference Manuals PN 6025005 and PN 6936508.

Analog Input Characteristics

- Resolution is 12 bits
- Input channels: 4 differential
- Input ranges: Switch Selectable 0 to +10 volts (unipolar), -5 to +5 volts (bipolar), -10 to +10 volts (bipolar)
- Digital coding: unipolar: binary; offset binary
- Safe input voltage: +/- 30 volts minimum input voltage (power on or off)
- Input current: +/- 4 milliamperes at maximum input voltage
- Common mode input range: +/- 11 volts maximum
- Common mode rejection: 72 db minimum ratio (signal within common mode range)
- Differential linearity error: +/- LSB maximum
- Differential linearity stability: +/- 5 ppm/degree C maximum, guaranteed monotonic
- Gain error: +/- 0.1% maximum between ranges: any range adjustable to zero
- Gain stability: +/- 32 ppm/degree C of FSR maximum
- Unipolar offset error: adjustable to zero
- Unipolar offset stability: +/- 24 ppm/degree C of FSR maximum
- Bipolar offset error: adjustable to zero
- Bipolar offset stability: +/- 24 ppm/degree C of FSR maximum
- Input resistance: 100 megohms minimum
- Input capacitance: 200 picofarads maximum measured at the distribution panel connector.
- Input leakage current: +/- 300 nanoamps maximum
- Settling time: channel acquisition: 20 microseconds maximum to within +/- 0.1% of the input value
- Conversion time: 35 microseconds maximum
- Throughput to memory: 15,000 conversions/second minimum
- A/D CE input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- A/D CO fanout: 10 LS TTL loads or 2 standard TTL loads

Analog Output Characteristics
- Resolution: 12 bits
- Output channels: 2
- Output ranges: switch selectable: 0 to +10 volts (unipolar) -5 to +5 volts (bipolar) -10 to +10 volts (bipolar)
- Digital coding: unipolar: binary; bipolar: offset binary
- Output load current: +/- 5 milliamps minimum
- Output load capacitance: 0.5 microfarads maximum for stability
- Output protection: protected for short to common
- Output impedance: 2 ohms maximum at distribution panel connector
- Differential linearity error: +/- 1/2 LSB maximum, guaranteed monotonic
- Gain error: +/- 0.1% maximum between ranges; any range adjustable to zero
- Gain stability: +/- 35 ppm/degree C of FSR maximum
- Unipolar offset error: +/- 3.25 millivolts maximum
- Unipolar offset stability: +/- 8 ppm/degree C of FSR maximum
- Bipolar offset error: adjustable to zero
- Bipolar offset stability: +/- 24 ppm/degree C of FSR maximum
- Power supply rejection: +/- 1/2 LSB maximum change in full-scale calibration
- Throughput from memory: 25,000 conversions/second maximum
- Dynamic characteristics for a +/- 10 volt step with less than +/- 5 milliamps and less than 1000 picofarad load
- Overshoot: +/- 1% of FSR maximum
- Settling time: 10 microseconds maximum to within +/- 0.1% FSR

Binary Input B10 Through B115
- Input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- Throughput to memory: 25,000 operations/second minimum
- BI Hold
- Input impedance: two LS TTL loads plus one 10 kilohm pull-up resistor
- BI STROBE
- Input impedance: one LS TTL load plus one 10 kilohm pull-up resistor
- BI CTS
- Fanout: 10 LS TTL loads or 2 standard TTL loads

Binary Output BO0 Through BO15
- Fanout: 28 LS TTL loads or 7 standard TTL loads
- Throughput from memory: 25,000 operations/second minimum
- BO GATE
- Input impedance: two LS TTL loads plus one 10 kilohm pull-up resistor
- BO CTS
- Input impedance: one LS TTL load plus one 10 kilohm pull-up resistor
- BO STROBE
- Fanout: 10 LS TTL loads or 2 standard TTL loads
32-Bit Timer Device (Counters 0 and 1)

COUNTER 0
- CLK 0 Frequency: 1.023 MHz
- RATE OUT
- Fanout: 10 LS TTL loads or 2 standard loads

COUNTER 1 DELAY OUT
- Fanout: 10 LS TTL loads or 2 standard TTL loads

16-Bit Counter Device
- Input impedance: one LS TTL load plus 10 kilohm pull-up resistor
- Input frequency: DC - 2 MHz (50% duty cycle)
- COUNT OUT
- Fanout: 10 LS TTL loads or 2 standard TTL loads

IBM PC DAC Adapter Distribution Panel
- Optional accessory for easy access to I/O signals, voltages and ground of the adapter
- Printed circuit board contains 4 barrier-type screw terminal strips for a total of 88 terminations
- Permanently attached shielded flat cable with 60-pin connector
- Cable and board assembly are housed in a metal enclosure with a slot for easy entry and exit of user cabling
- Meets FCC Class B requirements

Diagnostics
A diagnostic program to test the functionality and installation of the hardware is provided with each IBM PC DAC Adapter. The diagnostic requires a wrap connector which is provided with the adapter.

Power Requirements
- +5 volts +/- 5% at approximately 1 amp typical (1.5 amps maximum)

System Reference Voltage
- Output voltage: +10 volts
- Accuracy: +/- 1.2%
- Output load current: +/- 2 milliamps maximum
- Output load capacitance: 0.5 microfarads maximum at stability
- Output protection: protected for short to common
- Output impedance: 2 ohms maximum at distribution panel connector

Dimensions
- Length: 335.3mm (13.2in.)
- Height: 99.1mm (3.9in.)

Operating Environment
- Operating temperature: +15.6 to 32.2 C (+60 to +90 F)
- Relative humidity: 8 to 80% (non-condensing)
- Altitude: 2187m (7000 ft.) maximum

Setting Interrupts
Interrupts can be set via dios switches on the DAC Adapter. You have the ability to select interrupt request levels IRQ3 through IRQ7.

Requirements
The IBM PC DAC Adapter may be used with an IBM Personal Computer AT, the IBM PC XT, the IBM PC or the IBM Portable PC. The IBM DAC Adapter Programming Support requires the IBM Disk Operating System (DOS) 2.00 or higher and the IBM PC DAC Adapter. The IBM Personal Computer AT requires DOS 3.00.
IBM Data Acquisition and Control Adapter Programming Support

The IBM PC DAC Adapter Programming Support provides an easy-to-use interface for accessing the onboard and expansion analog and digital I/O capabilities of the adapter. The support operates under the IBM Disk Operating System (DOS) 2.00 or higher, and provides a device driver enabling the user of IBM PCs to configure the workstation. For development, there are sample programs in Interpretive BASICA, Compiled BASIC, Lattice C, FORTRAN 2.0 and IBM Professional FORTRAN.

Included are many useful programming support features through the use of function calls. Entire arrays of data that are input or output can be accomplished by one function call. Such functions are provided for analog input/output, binary input/output and for counter input. There are scanning functions that can scan a range of analog input channels, sampling functions that can be specified in samples per second, and full support of the expansion capabilities of the adapter. Examples in each of the supported languages for each of the functions are available in the DAC Adapter Programming Support manual.

Functions

When programming, three types of functions can be used:

- **Input Functions** collect input data and move it to memory
- **Output Functions** move data from memory to an external device
- **Utility Functions** control counters/timers and program execution

The software performs these functions according to rate or frequency. These functions may be further classified into:

- **SIMPLE FUNCTIONS.** Also called non-interactive functions. These execute only once.
- **MULTIPLE FUNCTIONS.** These iterative functions execute according to the number of times specified by a count argument in the function. A rate argument governs the rate of execution.
- **SCANNING FUNCTIONS.** Scans are sets of single inputs collected from a range of consecutively numbered channels. These samples are taken as close together as the device allows.

Function Names

Function names reflect the function type and the device involved. The first letter signifies the type of device:

- A analog device
- B binary device
- BIT a binary function that works with only a single bit of a binary word
- C counter device

The next two letters indicate the input/output:

- IN Input function
- OU Output function

The last letters show the rate types:

- S simple function
- M multiple function
- SC scanning instruction

Analog - Binary - Counters - Delay Functions

The analog I/O functions are:

- AINM Analog Input Multiple. Inputs values from the adapter, device and channel at the specified rate.
- AINS Analog Input Simple. Inputs a single value from the adapter, device and channel.
- AINSC Analog Input Scan. Inputs values from multiple channels on the input device at the rate specified.
- AOUM Analog Output Multiple. Outputs a range of variables to the adapter, device and channel at the rate specified.
- AOUS Analog Output Simple. Outputs a single variable to the adapter, device and channel.
The Binary I/O Functions are:

- **BINM**: Binary Multiple. Inputs selected states of the binary input word at the rate specified.
- **BINS**: Binary Input Simple. Inputs a single binary word from the adapter and device.
- **BITINS**: Binary Input Bit Simple. Inputs the state of a bit in the binary input word.
- **BITOUS**: Binary Bit Output Simple. Outputs the state of a bit in the binary output word.
- **BOUM**: Binary Output Multiple. Outputs selected binary state variables at the rate specified.
- **BOUS**: Binary Output Simple. Outputs the contents of the data variable as a binary word.

The Counter Functions are:

- **CINM**: Counter Input Multiple. Reads counter values from the adapter at the rate specified.
- **CINS**: Counter Input Simple. Reads the adapter counter value.
- **CSET**: Counter Set. Initializes the adapter counter.

The Delay Functions is:

- **DELAY**: Delay execution. Delay application program execution.

Performance

The user can extend device driver performance through the use of the extended execution mode operation. The following data rates can be achieved:

- **AINM**: 15 K Samples/Second
- **AOUM, BINM, BOUM**: 17 K Samples/Second
- **CINM**: 12 K Samples/Second

Configuration Requirements

The minimum configuration allowed includes:

**Machine Requirements**

- An IBM PC with a minimum of 64 KB of memory (BASIC, C, FORTRAN compilers require additional)
- At least one 320 KB diskette drive
- An IBM monitor
- The DAC Adapter
- Any cables necessary to connect the adapter to the device it is to interface.

**Programming Requirements**

- IBM PC Disk Operating System, Version 2.00 or later
- IBM PC DAC Adapter Programming Support
- Any of the following optional languages:
  - IBM PC BASICA Interpreter
  - IBM PC BASIC Compiler
  - IBM FORTRAN Compiler Version 2.00
  - IBM PC Professional FORTRAN Compiler
  - C Lattice Compiler Version 2.0

Adapter Dependent Information

The adapter switches are set to give the computer and software the following values:

- Interrupt level
- Adapter number
- Analog input and output ranges.
Setting interrupts

Unlike some Personal Computer peripherals, the DAC Adapter and Programming Support use a shared interrupt system. While you can use vectored and shared interrupt devices in the same system, no vectored interrupt can have the same priority as that of the adapter. The adapter can use interrupt request levels IRQ3 through IRQ7.

Assigning adapter numbers

Each adapter has a number that identifies it to the system. This is the adapter number. This number is included in the argument list of any function call that uses the adapter. Before installing the adapter, set the switches to assign the number (0-3).

Setting analog ranges

Adapter switch governs the voltage range of the analog input and analog output devices. To accurately compute analog I/O values, you must know the full-scale range of your analog I/O hardware. These ranges are:

-5 to 5 volts
-10 to 10 volts
0 to 10 volts

BASIC Sample Program

The following paragraphs contain the sample program description and the Interpretive BASICA code to demonstrate the programming of selected I/O function calls supported by the DAC Adapter Programming Support software. Each section of the BASIC sample program illustrates the use of one analog I/O function. The program begins with a header and concludes with brief error-handling and data-plotting routines. The header executes first. The program pauses, displaying a prompt until you press the Enter key. Then Section 1 runs. Sections 2 through 5 work in the same way.

Requirements

The BASIC sample program requires the following:

- Analog input and analog output ranges set to -5 to +5 volts
- I/O address for the adapter set to 0

- Color monitor for Sections 4 and 5
- Test circuit
- Distribution Panel.

Wiring the Test Circuit

The sample program requires a test circuit. You use this to run the sample program without connections to external devices. The circuit provides analog input signals which it derives from internal voltages on the adapter. It also allows you to see, by varying the intensity of an LED, the effect of changing analog output voltage.

The test circuit components required are:

- A 100 K ohm potentiometer.
- A standard Light Emitting Diode (LED) with a 10 K ohm resistor ballast.
- Three clip leads.
- 24 inches of 17 gauge insulated, stranded wire. Use this wire to make six 4-inch jumpers, stripped 1/2-inch at each end.

Wire the test circuit to the DAC Adapter Distribution Panel as specified in the following table.

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/A 0</td>
<td>Pot end</td>
</tr>
<tr>
<td>D/A 0</td>
<td>LED + (red)</td>
</tr>
<tr>
<td>D/A 1</td>
<td>Pot end</td>
</tr>
<tr>
<td>AGND</td>
<td>LED - (black)</td>
</tr>
<tr>
<td>AGND</td>
<td>A/D 0 -</td>
</tr>
<tr>
<td>AGND</td>
<td>A/D 1 -</td>
</tr>
<tr>
<td>A/D 0</td>
<td>BO8</td>
</tr>
<tr>
<td>A/D 1</td>
<td>Pot wiper</td>
</tr>
</tbody>
</table>

Adding the Header

The BASICA program requires a header supplied on the DAC Adapter Programming Support Distribution Diskette. This header is a pre-written section of BASICA code. It must appear at the beginning of each program. The header code is not included in the sample program listing.

After the header executes, every DAC Adapter software function is accessible as a real variable.
Global Variables

The sample program sections use global variables of their own. You must initialize them as shown here:

120 ADAPTS = 0
140 'use on-board analog I/O device
150 DEVICE% = 9
160 'and dimension a 640-point array
170 'to be used in sections 3, 4, and 5
180 DIM RAWDATA% (640)

For the purposes of these samples, assign address 0 to the adapter.

All of the sample program sections deal with the on-board analog I/O device. Because of this, the programs set DEVICE% = 9. They also DIMension an integer array that will hold 640 data points. The number 640 is somewhat arbitrary. The simple data plotting routine used by several of the samples has a 640-point horizontal resolution. In actual practice, you will probably need to dimension all types of arrays at various points in the program. To provide a clear sample, these programs use a single array, named RAWDATA%. This array stores all data read and written by multiple analog I/O functions (AINM, AINSC, and AOUM).

Section 1: A Simple Analog Output from Two Channels

This section uses the function AOUS (Analog Output Simple) to:

- output voltages of +5 Volts to analog output channel 1
- output –5 Volts to analog output channel 0.

These voltages serve as the source of the analog signals that are used in the subsequent sections.

The first step is to assign values for the remaining arguments of AOUS (ADAPTS% and DEVICE%, remember, have been assigned already).

1070 CTRL$% = 0
1080 STATS = 0
1090 LOVOLTS% = 0
1100 HI VOLTS% = 4095
1110 LOCHANS = 0
1120 HI CHANS = 1

The first call to AOUS sets the output voltage at D/A channel 0 (LOCHANS%) to –5 Volts (LOVOLTS%).

1140 CALL AOUS (ADAPTS, DEVICES, LOCHANS,
CTRL$, LOVOLTS, STATS)

Note: You must set CTRL$% (the expansion device control argument) to 0 when accessing the on-board devices.

Following this call (and all calls to the adapter) functions take the time to execute a test and conditional jump to a brief error-handling routine.

The program sets the execution status variable (STAT%) to 0 before each call. Immediately after each call, it tests STAT%. If STAT% is non-zero, it then sets the variable LNUM% equal to the line number in which the error occurred. The error handler then comes into play, as shown below:

1160 IF STAT% = 0 THEN LNUM% = 1140: GOTO 6000

The error handler looks like this:

6000 '***** Error handler begins here *****
6010 'if an error occurs, print message, number, and exit
6020 PRINT "Execution Error %", STAT%, "line number ". LNUM%
6030 PRINT "Program terminated"
6050 END

If an error occurs, the line and error numbers print, enabling you to determine from the status code what went wrong.

The program then continues, setting D/A channel 1 (HI CHANS%) to +5 Volts (HI VOLTS%).

1180 CALL AOUS (ADAPTS, DEVICES, HI CHANS, CTRL$, HI VOLTS, STATS)

After testing STAT% again, the program prints a message on the screen and waits for you to press the Enter key.

1220 PRINT "Output voltages set."
1230 PRINT "D/A 0 terminal is –5 Volts"
1240 PRINT "D/A 1 terminal is +5 Volts"
1250 PRINT "Press ENTER to run Section 2", C$.
1260 INPUT "PRESS ENTER TO RUM SECTION 2", C$.
1270 CLS
1280 PRINT "Section 2 begins."

When you press the Enter key, the screen clears and Section 2 begins.
Section 2: A Simple Analog Input from a Single Channel

Section 2 uses AINS (Analog input Simple) in a loop to continuously read channel 1 of the on-board analog input device of adapter 0. Channel 1 connects through the potentiometer to +/- 5 volts. By twisting the shaft of the potentiometer, you can modulate the amount of the +/-5 volt signal that reaches analog input channel 1.

As in the section 1, the first step is to assign values for the arguments of the function.

2000 CHANNES = 1
2010 CIRS = 0
2020 STATE = 0

Note: Economical programming standards probably dictate a simple reuse of the variable HICHAN% for the channel number argument. However, the program explicitly re-specifies a CHANNEL% variable here; this emphasizes that all variables used by the CALL statement must be assigned before the program performs the call.

The actual sampling takes place under the control of a WHILE...WEND structure. This takes a sample, displays it as a voltage, and checks for a halt request from the keyboard. As long as no key has been pressed, the adapter samples channel 1 continuously.

2100 HALT% = **
2110 LOCATE 23, 1
2120 PRINT “Press any key to stop sampling.”
2130 WHILE HALT% = **
2140 ‘get a sample’
2150 CALL AINS (ADAPTS, DEVICES, CHANNELS, CIRS, VS, STATE)
2160 ‘if status non-zero, set line and go to error handler’
2170 IF STATUS = 0 THEN LINE% 2150 0010 0000
2180 ‘convert raw A/D value to volts’
2190 VOLTS = VS/4096
2200 ‘print time of day’
2210 LOCATE 1, 6
2220 PRINT TIMES
2230 ‘print voltage’
2240 LOCATE 4, 1, 0
2250 PRINT USING “Channel 1 input voltage is %%: %0.3f VOLS
2260 LOCATE 4, 27, 0
2270 ‘test for halt request’
2280 HALT% = INKEYS
2290 WEND

The conversion-to-volts equation in line 2190 converts the 12-bit values (in the range 0 to 4095) returned by the analog input device. It becomes a representation of input voltage in the range +/- 5 Volts. If the analog input system were configured to a different range, you would have to change this equation accordingly.

The general form of this equation is:

\[ \text{VOLTAGE} = \left(\frac{\text{CODE} \times \text{RANGE}}{2^n}\right) - \text{OFFSET} \]

where

- CODE is the raw A/D value.
- RANGE is the A/D.
- Reference voltage span is in volts.
- n is the number of bits of resolution of the A/D converter (exponent of 2).
- OFFSET is the negative offset of the A/D range. A 0 to 10V range, for example, uses the equation VOLS = (CODE/4095.6) (there is no negative offset). Similarly, if you were using an expansion device of different resolution, you would have to change n to reflect the resolution of that device.

During execution of this program section you turn the shaft of the potentiometer from one stop to the next, varying the fraction of the +/- 5V signal supplied to analog input channel 1. The changing voltage is displayed on the screen.

Section 3: Multiple Analog Inputs From a Single Channel

This program uses AINM to input 640 analog values into a one-dimensional array. Unlike the single-sample (AINS) loop used in Section 2, AINM acquires signals at precisely-timed intervals. Since the input signal measured (the voltage modulated by the pot) does not fluctuate very rapidly, you can use a slow sampling rate of 200 samples per-second. The first lines of this program assign values to the arguments of AINM. Remember, the program has already DIMensioned the RAWDATA% array to include 640 elements.

3000 CHANNELS = 1
3010 CIRS = 0
3020 MODE = 0
3030 STOPS = 0
3100 COUNT = 640
3110 RATE = 200
3120 STATE = 0
In this program, COUNT and RATE are real, rather than integer, variables. They are the only non-integer variables for arguments to the functions.

The variable COUNT must not exceed the amount of storage available in the target array. AIMN acquires count sample, so a statement of the form:

```
DIA ARRAYS (COUNT = 1)
```

allocates the correct number of array elements.

Often, timed sampling of this sort works in conjunction with a triggering structure. Otherwise, you run a risk that the function will sample while no data is being generated. In this example, pressing the Enter key triggers the function.

```
3110 INPUT "Press ENTER to begin sampling ", CS
3120 \{print and print "Sampling" message, then do AIMN
3130 BEEP - CLS
3140 PRINT "Sampling analog input channel 1...
3150 CALL AIMN (ADAPT, DEVICES, CHANNELS, CTRLS,
3160 \{OVES, STOPS, COUNT, RATE, RANGE1(0), STATS)

When the sampling is started the shaft of the potentiometer back and forth to vary the input signal. Once the program collects 640 data points, it jumps to a graphing routine. The routine plots the data, then waits for you to press Enter before moving to the next section.

```
3210 \{graph points in RAMARRAYS
3220 GOSUB 10000
3230 \n```

Section 4: An Analog Input Scan of Multiple Channels

Section 4 uses the function AINSC (Analog Input Scan) to input 320 analog values from each of two channels into a two-dimensional array. This section is in most respects identical to Section 3. It first executes a brief subroutine that outputs a binary low voltage (around 0 volts) to analog input channel 0. This simply provides a second data value for the scan to record. If connected to analog input channel 0, it "floats" to an unpredictable value. By driving it low with a binary output, you guarantee that it returns a constant value near 0 volts.

A simple GOSUB statement in line 4050 jumps to the following subroutine.

```
DIA ARRAYS (COUNT = COUNT - 1)
```

This subroutine sets up the required arguments for BITOUI (binary Bit Output Simple); it sets the value of bit 0 to 0. Note that you must use a different variable name, as well as a different number, to specify the device. Simply re-assigning DEVICE%, causes subsequent analog I/O calls to attempt to access device 8. This condition results in an error indication being returned in the status variable.

Next, the program assigns values to the arguments of AINSC. Note that there are two channel arguments: CL%, specifying the low channel (the first channel in the scan) and CH%, specifying the high channel (the last channel in the scan). For each scan specified by the COUNT, AINSC inputs a sample from every channel in the range CL% to CH%, starting with CL% and ending after CH%. In this case, each scan consists of an input from channel 0 followed by an input from channel 1.

```
4080 CLS = 0
4090 CLS = 1
4100 CTRL= 0
4110 MODES = 0
4120 STORE = 0
```

Since each scan can acquire multiple samples, be sure to reserve enough array space for the data by a scanning input function such as AINSC. When DIMensioning arrays filled from scans, you may wish to use a statement of the type:

```
DIA ARRAYS (COUNT = COUNT - (CH% - CL% + 1))
```

In the case of this section, you'll find it most convenient to simply reduce the count.

```
4110 COUNT = 320
4120 RATE = 200
4130 STATE = 0
4140 \```

This section uses a triggering structure identical to the one used in Section 3. The only difference is in the data acquisition function itself.
Section 5: Multiple Analog Outputs From a Single Channel

The final section uses the function AOUM (Analog Output Multiple) to reverse the data acquisition process. This outputs the contents of an array (RAWDATA), via D/A channel 1, as a varying voltage — an analog signal. When you run this section, the contents of RAWDATA are output to analog output channel 0. The LED, connected from D/A channel 0 to analog ground (AGND), glows with a varying intensity as the output voltage changes.

The arguments of AOUM first assume these values:

5070 CHANNELS = 1
5080 CTRLS = 0
5090 MODES = 0
5100 STORES = 0
5110 COUNT = 440
5120 RATE = 200
5130 STATS = 0

Now print a message and call AOUM.

5160 PRINT "Outputting data acquired in Section 4."
5170 ' and do AOUM
5190 CALL AOUM (ADAPTS, DEVICES, CHANNELS, CTRLS,
MODES, STORES, COUNT, RATE, RAWDATA(0), STATE)

BASIC Example Program Listing

1 ' ! PROGRAM: BASIC Example Program for the
2 ' ! IBM Data Acquisition and Control Adapter
3 ' 100 'initialize global variables used in all sections
110 ' adapter number 0
120 ' ADAPTS = 0
140 ' use on-board analog 1/0 device
150 ' DEVICES = 9

IBM PC Data Acquisition and Control Adapter and Software
IBM PC Data Acquisition and Control Adapter and Software
1000 ' Arguments: LENENTS = number of points to plot
10010 ' RANOATRAS( ) = array to be plotted
10040 ' Affects: The screen is left in mode 2. All
10050 ' variables ending in CRT are reserved.
10060 ' 10070 ' This subroutine plots the elements of the array
10080 ' RANDATRAS( ) on a color display. The elements are
10090 ' windowed to the range specified by MAXVAL, MINVAL, and
10100 ' screen size. The number of points plotted must be
10110 ' passed as an argument in the variable LENGTHS.
10120 ' The screen clears upon each execution of SIMPLOT.
10130
10200 ' 20000 MAXVAL CRT = 2000 ' maximum value to be plotted
10205 ' LENGTHS = 640
10210 ' MINVAL CRT = 0 ' minimum value to be plotted
10220 ' SCREEN 2
10230 ' 20060 CLS
10250 ' FOR 1 GRF = 0 TO LENGTHS
10260 ' NEXT (GRFS, 0, MINVAL, CRT) (LENGTHS - 1, MAXVAL, CRT)
10280 ' FOR I GRFS = 0 TO LENGTHS
10290 ' NEXT (GRFS, 0, MINVAL, CRT)
10300 ' PLOT (1 GRFS, RANDATRAS1GRFS) - 1
10310 ' NEXT (GRFS, 0, MINVAL, CRT)
10320 ' RETURN
10330 ' RETURN

IBM PC Data Acquisition and Control Adapter and Software
APPENDIX G. PROJECT BILL OF MATERIALS
Table G-1. Project Bill of Materials

<table>
<thead>
<tr>
<th>ITEMS</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solar collector panel</td>
<td></td>
</tr>
<tr>
<td>1 panel</td>
<td>30.00</td>
</tr>
<tr>
<td>2 lamps</td>
<td>20.00</td>
</tr>
<tr>
<td>Distribution unit</td>
<td></td>
</tr>
<tr>
<td>3 Solenoids</td>
<td>75.00</td>
</tr>
<tr>
<td>2 Pumps</td>
<td>35.00</td>
</tr>
<tr>
<td>1 2-quart plastic bottle</td>
<td>15.00</td>
</tr>
<tr>
<td>ethylene glycol fluid (1 gallon)</td>
<td>5.00</td>
</tr>
<tr>
<td>tygon tubing (20 yards, 1/4&quot; ID)</td>
<td>5.00</td>
</tr>
<tr>
<td>piece of board (14&quot; X 20&quot; X 1/2&quot;)</td>
<td>1.00</td>
</tr>
<tr>
<td>screws, wires, and clamps</td>
<td>3.00</td>
</tr>
<tr>
<td>Heat storage unit</td>
<td></td>
</tr>
<tr>
<td>36 quart Igloo picnic cooler</td>
<td>15.00</td>
</tr>
<tr>
<td>5 bricks</td>
<td>2.00</td>
</tr>
<tr>
<td>Copper/Aluminum heat exchanger (see Figure 5)</td>
<td>20.00</td>
</tr>
<tr>
<td>Cooling system</td>
<td></td>
</tr>
<tr>
<td>12 yards 1/2&quot; id Copper tubing</td>
<td>5.00</td>
</tr>
<tr>
<td>48 quart Igloo picnic cooler</td>
<td>25.00</td>
</tr>
<tr>
<td>fan</td>
<td>5.00</td>
</tr>
<tr>
<td>House</td>
<td></td>
</tr>
<tr>
<td>Barn</td>
<td>10.00</td>
</tr>
<tr>
<td>Styrofoam (36&quot; X 36&quot; X 3/4&quot;)</td>
<td>1.00</td>
</tr>
<tr>
<td>Copper/Aluminum heat exchanger (see Figure 2)</td>
<td>20.00</td>
</tr>
<tr>
<td>corking</td>
<td>1.00</td>
</tr>
<tr>
<td>Computer override switch (Figure 6)</td>
<td></td>
</tr>
<tr>
<td>switch</td>
<td>1.00</td>
</tr>
<tr>
<td>casing box</td>
<td>1.50</td>
</tr>
<tr>
<td>fuse</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Table G-1. Continued (Project Bill of Materials)

<table>
<thead>
<tr>
<th>Component</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 grounded-outlets</td>
<td>2.00</td>
</tr>
<tr>
<td>optoisolator</td>
<td>10.00</td>
</tr>
<tr>
<td><strong>Power Supply</strong></td>
<td></td>
</tr>
<tr>
<td>3 transformers (18 VAC)</td>
<td>15.00</td>
</tr>
<tr>
<td>2 2200uF capacitors</td>
<td>4.50</td>
</tr>
<tr>
<td>switch</td>
<td>2.00</td>
</tr>
<tr>
<td>2 full-wave bridges</td>
<td>1.00</td>
</tr>
<tr>
<td>1 5V regulator</td>
<td>3.00</td>
</tr>
<tr>
<td>1 12V regulator</td>
<td>3.00</td>
</tr>
<tr>
<td>wires as needed</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>Interface box</strong></td>
<td></td>
</tr>
<tr>
<td>6 5V relays</td>
<td>18.00</td>
</tr>
<tr>
<td>6 optoisolators</td>
<td>6.00</td>
</tr>
<tr>
<td>1 8055 amplifier chip (ECG 2021)</td>
<td>3.00</td>
</tr>
<tr>
<td>1 ribbon cables</td>
<td>3.00</td>
</tr>
<tr>
<td>Housing cabinet</td>
<td>3.00</td>
</tr>
<tr>
<td>12 resistors</td>
<td>1.00</td>
</tr>
<tr>
<td>6 LEDs</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Wire connection network</strong></td>
<td></td>
</tr>
<tr>
<td>3 ribbon cables with a 16 pin end</td>
<td>9.00</td>
</tr>
<tr>
<td>2 LEDs</td>
<td>0.25</td>
</tr>
<tr>
<td>wires as needed</td>
<td>5.00</td>
</tr>
<tr>
<td><strong>Sensing system</strong></td>
<td></td>
</tr>
<tr>
<td>1 photoresistor</td>
<td>1.00</td>
</tr>
<tr>
<td>3 thermisters</td>
<td>1.50</td>
</tr>
<tr>
<td>4 thermometers (for calibration only)</td>
<td>20.00</td>
</tr>
</tbody>
</table>