USUS NewsLetter
Nov/Dec 1989
Copyright 1989 USUS, Inc.
All the News that Fits, We Print
William D. Smith, Editor
Volume 3
Number 8

## From the Editor
by William D. Smith

Well now what do you know? I have just finished this, my last scheduled NewsLetter. A little late as usual.

As of this time no one has come forward to take over this job (or for that matter the Administrator's job). That means, the next NewsLetter will arrive who knows when in the future.

I have enjoyed doing the NewsLetter and will continue to submit articles (units and programs). At this time I have two possible articles from **David Craig** (not in electronic format yet) about the Macintosh and one from **Felix Bearden** (arrived after the NewsLetter was layed out) reviewing Insight Window Designer and KSAM. If you wish more information on what is involved, you may contact me at CIS: 73007,173 or (619) 941-4452 (after 10am please).

I hope you all had a Happy Thanksgiving and wish you a Merry Christmas and prosperous New Year.

## Administrator Says
by Hays Busch

First, a couple of quick bits of information.

FOG (First Osborne Group) offers a disk format conversion service that can handle over 200 different formats. They charge $15 for the first disk conversion on an order, and $5 each for each added disk of the same format on the same order. Obviously, many markets have commercial outfits that provide this service, but if you are stuck for a format, FOG may have the ability to help. (There are some disk formats on their list I've never even heard of before!!) Any member wanting a copy of the write up on this service needs only to drop me a note and I'll mail it to you.

SIR TECH offers user groups a 20 percent discount on their game software. They have eight games for Apple and IBM, and a few for Macintosh, Commodore and Atari. Prices range from $9.95 to $63.96. Titles include "Proving Grounds", "Knight of Diamonds", "Legacy of Llylgamyn", "The Return of Werdna" and "Heart of the Maelstrom" in the Wizzardy series. (NOTE: I have not seen any of these work so have no comments as a reviewer.) If you want a copy of their brochure, drop me a note. Takes ten orders from the group to qualify for the discount. We'll see if there is enough interest based on your requests for the brochure.

---

SO NOW IT'S GOODBYE! It's been three years for me as Administrator and it's time for me give someone else a chance!

The job is now down to a "quiet roar" and takes only 1 to 3 days a week at most. That's if it is done the way I do it. Someone else could be a lot faster than I am. I know the job can be split into at least three jobs without much loss of efficiency and if this could happen it would be about a day a week for each job.

Our Editor mentioned I was "stepping down" in the September/October issue, but so far no one has called me to ask about taking over. Someone needs to! USUS can not run without someone taking care of the details you all have come to expect from the organization.

Also, someone with the ability to edit and produce the artwork for the NewsLetter needs to take over from William Smith. Without a NewsLetter, much of what USUS means to members will be lost. The NewsLetter need not be produced on a Macintosh and Laser printer...which is how William does it... but hopefully it will be the product of a desktop publishing effort.

**Treasurer's Report** (September 1989)
by Robert E. Clark, Treasurer

| | | |
|---|---|---|
| Bank Balance | $5740.00 | 08-31-89 |
| Income - September 1989 | | |
| Dues: | | (new/renew) |
| Student | 25.00 | 1/0 |
| General | 280.00 | 1/7 |
| Professional | 100.00 | 0/1 |
| Institutional | 0.00 | 0/0 |
| Other Income: | | |
| Library fees | 13.00 | |
| CIS | 40.18 | |
| Total Income: | $458.18 | |
| Expenses - September 1989 | | |
| Administrator: | | |
| CIS | 79.01 | |
| Telephone | 5.53 | |
| Postage | 7.91 | |
| Supplies | 11.20 | |
| Equipment | 53.00 | |
| Repairs | 317.00 | |
| Other: | | |
| La Jolla PO box rent | 36.00 | |
| Refund | 5.00 | |
| Bank charge | 1.00 | |
| Total Expenses | $515.65 | |
| Bank Balance | $5,682.53 | 09-30-89 |

**Treasurer's Report** (October 1989)
by Robert E. Clark, Treasurer

| | | |
|---|---|---|
| Bank Balance | $5,682.53 | 09-30-89 |
| Income - October 1989 | | |
| Dues: | | (new/renew) |
| Student | 50.00 | 0/2 |
| General | 1080.00 | 0/30 |
| Professional | 100.00 | 0/1 |
| Institutional | 0.00 | 0/0 |
| Other Income: | | |
| Library fees | 6.00 | |
| CIS | 36.84 | |
| Donations | 15.00 | |
| Publications | 14.50 | |
| Total Income: | $1,302.34 | |
| Expenses - October 1989 | | |
| Administrator: | | |
| Telephone | 13.88 | |
| Photocopies | 25.52 | |
| Postage | 34.57 | |
| Other: | | |
| Mail from La Jolla | 1.05 | |
| Library Distributor | 5.00 | |
| Stamps | 10.00 | |
| Bank charge | 1.00 | |
| Total Expenses | $91.02 | |
| Bank Balance | $6,893.85 | 10-31-89 |

To all of you who helped me as Administrator, heartfelt thanks... **Frank Lawyer, William Smith, Bob Clark, Robert Geeslin, Alex Kleider** and **Verlene Bonham**. To **Jon Nevins** and **Bob Clark**, thanks for PowerTools. To **Sam'l Bassett**, thanks for the early efforts on the NewsLetter. To **Harry Baya** and **William Smith**, thanks for taking over MUSUS.

To members who helped with the SW Library: **William Reed, Gordy Kastner, David Craig, P.D. Terry, Carl Helmers, Henry Baumgarten, Bob Clark, Ken Hamai** and last but not least **Frank Lawyer**, my thanks.

And to all USUS members, and anyone I forgot to mention above, thanks for your support of the group for the past three or more years. I have been impressed by how many original USUS members are still with us and also how many of our newest members are renewing. It's gotten so I can recognize most of you by name even tho we may not have met.

I've been a member of USUS since sometime in 1980. UCSD Pascal was the only language I ever learned. The p-System is the only operating system I ever used. A fellow by the name of Chuck Howerton and I wrote a complete retail inventory program for my ski shops. It was written in Version II and ran on DEC LSI-11s. It served the business well until last year when sales volume simply outgrew the machines and the programs. (And I'd hate to tell you what it cost the business to replace the old stuff with spiffy new Hewlett Packard machine that is on-line to all stores and which collects data 24 hours a day from wand reading cash registers. Put another way, storage media went from 8-inch

SSSD floppies to 800 MB of hard disk storage. Quite a jump!)

But as we all know, nothing holds still in the computer environment. So it's time for me to step aside. I'll work with whoever comes forth to take the job on. That way there will be a smooth transition. Being Administrator is an important and interesting responsibility in USUS, and its one of the jobs that is vital for the continuation of USUS.

Any takers???

**Deque Module**

By Peter M. Perchansky
412-1 Springside, Drive East
Shillington, PA 19607
CompuServe: 71301,344
FidoNet: 1:273/101

The following *(pages 4 thru 11)* is my implementation of a generic deque using JPI's TopSpeed Modula-2 (version 1.17). A deque is a form of double-ended queue (or stack depending on your point of view). The deque abstract data type allows you to perform queue and stack operations.

The queue operations are based on the following: The front of the queue is the root node. Enqueueing data will place the node at the rear of the queue. Dequeueing (serving) data will remove the node from the front of the queue.

The stack operations are based on the following: The top of the stack is the end node. Pushing data will place the node at the rear (top) of the stack. Popping data will remove data from the rear (top) of the stack.

This library module is generic. Any data type (both standard and user-defined) can be operated on in the deque. You can even mix and match data types within the deque; however, you are responsible for keeping track of what types are in the deque.

Future versions of this module will include procedures to compare deques, append one deque to another deque, search for elements within the deque, etc.

Please feel free to contact me via postal mail, CompuServe, FidoNet, or Interlink (language or Pascal Conference <they are working on adding a Modula-2 Conference>) if you have any questions or comments.

**WDS Terminal Typed I/O Unit**

By William D. Smith

Last NewsLetter I presented my terminal I/O unit which allowed input and output of strings and characters. This unit *(code startting on page 13)* uses that unit to read and write a formatted string from and to the terminal. It uses the string ops unit to do the formatting. At this time there is support for four data types: integers, dates, time and YesNo (a two, three or four way switch). There is a get procedure (beginning with "`G_`") and a write procedure (beginning with "`W_`") for each of these four types. There are also two miscellaneous procedures for informing the user of current progress: `InForm` and `W_Dot`. `InForm` writes a message near the bottom of the screen (such as "Reading 100 records") and positions the cursor for `W_Dot`. `W_Dot` writes one dot on the current line up to fifty. It then erases the line and starts again. The line begins with the last dot written (eg. "< 50>....."). `InForm` should be called once before a each loop with the message and once after the last loop with null values for its parameters. This last call cleans up the screen. `W_Dot` is called once each time through a loop. The counter is initialized before the first loop (and maybe re-initialized before any of the other loops) and is incremented by `W_Dot`.

Each of the write procedures clear the field (Len spaces or the defined amount for date and time types) if passed a null value.

Input is restricted by the number of characters which are allowed to be entered and then validated after the entry is accepted.

```
(*=============================================================================
  =  Program Id:      PMPDEQUE                                                =
  =  Programmer:      Peter M. Perchansky                                     =
  =  Purpose:         Export procedures for dealing with deques.              =
  =  Date Written:    10-29-89                                                =
  =============================================================================*)

DEFINITION MODULE PMPDeque;

(*------------- Procedures from JPI's TopSpeed Modula II ------------*)
FROM SYSTEM IMPORT BYTE;
(*-------------------------------------------------------------------*)

TYPE  Deques;          (* opaque type defined in implementation module *)

(*  A deque is a circular queue allowing for both queue and stack     *)
(*  operations.  This module is generic, allowing for operations on   *)
(*  any data type.                                                    *)

(*  The deque must be initialized to NIL by calling Create (deque).   *)
(*  Create (deque) should only be called once.  DestroyDeque should   *)
(*  be called when the deque is no longer needed.                     *)

(*  Procedures to initialize/remove the deque:                        *)
(*            CreateDeque  - set deque to NIL                         *)
(*            DestroyDeque - remove deque from memory                 *)

(*  Functions providing status information on the deque:              *)
(*            Empty        - is deque NIL                             *)
(*            Full         - is deque full                            *)
(*            DequeLength  - number of nodes in deque                 *)
(*            DequePos     - node number containing data (if found)   *)
(*            InDeque      - is data in deque                         *)

(*  Procedures to enter data in the deque:                            *)
(*    <queue> Enqueue      - place data in rear of deque              *)
(*    <stack> Push         - place data in rear of deque              *)

(*  Procedures to remove data from the deque:                         *)
(*    <queue> Dequeue      - remove data from the front of the deque  *)
(*    <stack> Pop          - remove data from the rear of the deque   *)
(*            Serve        - remove nth item in the deque             *)

(*  Procedures to show data in the deque:                             *)
(*    <queue> Front        - show data from the front of the deque    *)
(*    <queue> Back         - show data from the rear of the deque     *)
(*    <stack> Top          - show data at top of the deque            *)
(*            Retrieve     - show data from nth item in the deque     *)

PROCEDURE CreateDeque  (VAR deque : Deques);
(* Creates the deque by making it NIL.  This procedure must be called *)
(* prior to performing operations on the deque.                       *)

PROCEDURE Empty        (deque : Deques) : BOOLEAN;
(* Return TRUE if the deque is NIL                                    *)

PROCEDURE DestroyDeque (VAR deque : Deques);
(* Destroys the deque by deallocating deque nodes and contents.       *)
(* DestroyDeque should be called when the deque is no longer needed   *)
(* in memory.                                                         *)

PROCEDURE Full         (size : CARDINAL) : BOOLEAN;
(* Return TRUE if there is no Available memory for another deque node *)

PROCEDURE DequeLength  (deque : Deques) : CARDINAL;
(* Returns the number of nodes in the deque                           *)
```

```
PROCEDURE DequePos      (data : ARRAY OF BYTE; deque : Deques) : CARDINAL;
(* Returns the number of the node containing the data (if found).     *)
(* Returns 0 if not found.  The front of the deque is treated as node *)
(* number one.                                                        *)

PROCEDURE InDeque       (data : ARRAY OF BYTE; deque : Deques) : BOOLEAN;
(* Returns TRUE if data is found in deque.                            *)

PROCEDURE Enqueue       (    data : ARRAY OF BYTE;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place data in node at the back of the deque.  Ok will be set to    *)
(* FALSE under the following conditions:  Deque is full.              *)

PROCEDURE Push          (     data : ARRAY OF BYTE;
                          VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place data in node at the back of the deque.  Ok will be set to    *)
(* FALSE under the following conditions:  Deque is full.              *)

PROCEDURE Dequeue       (VAR data : ARRAY OF BYTE;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the front of the deque into data.      *)
(* Remove the front node of the deque.  Ok is set to FALSE upon the   *)
(* following conditions:  deque is empty, size of data does not equal *)
(* size of contents.                                                  *)

PROCEDURE Pop           (VAR data : ARRAY OF BYTE;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back of the deque into data.       *)
(* Remove the back node of the deque.  Ok is set to FALSE upon the    *)
(* following conditions:  deque is empty, size of data does not equal *)
(* size of contents.                                                  *)

PROCEDURE Serve         (VAR data : ARRAY OF BYTE;
                             nthItem : CARDINAL;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from nth node of the deque into data.  Remove the   *)
(* nth node of the deque.  Ok is set to FALSE upon the following      *)
(* conditions:  deque is empty, size of data does not equal size of   *)
(* contents, nthItem is greater than number of nodes in the deque.    *)

PROCEDURE Front         (VAR data : ARRAY OF BYTE;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the front of the deque into data.      *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

PROCEDURE Back          (VAR data : ARRAY OF BYTE;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back of the deque into data.       *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

PROCEDURE Top           (VAR data : ARRAY OF BYTE;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back (top) of the deque into data. *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

PROCEDURE Retrieve      (VAR data : ARRAY OF BYTE;  nthItem : CARDINAL;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from nth node of the deque into data.  Ok is set to *)
(* FALSE upon the following conditions:  deque is empty, size of data *)
(* does not equal size of contents.  nthItem is greater than number   *)
(* of nodes in the deque.                                             *)

END PMPDeque.
```

```
(*=============================================================================
  =  Program Id:      PMPDEQUE                                                =
  =  Programmer:      Peter M. Perchansky                                     =
  =  Purpose:         Export procedures for dealing deques.                   =
  =  Date Written:    10-29-89                                                =
  =============================================================================*)

IMPLEMENTATION MODULE PMPDeque;

(*------------- Procedures from JPI's TopSpeed Modula II -------------*)
FROM Lib       IMPORT Compare, Move;
FROM SYSTEM    IMPORT ADDRESS, BYTE, TSIZE;
FROM Storage   IMPORT ALLOCATE, Available, DEALLOCATE;
(*--------------------------------------------------------------------*)

TYPE
    Deques         = POINTER TO DequeNodes;   (* opaque type defined *)

    DequeNodes     = RECORD                   (* node *)
                       contents : ADDRESS;    (* contents of node *)
                       size     : CARDINAL;   (* size of contents *)
                       next     : Deques;     (* pointer to next node *)
                       previous : Deques;     (* pointer to prev node *)
                     END;

(*--------------------------------------------------------------------*)
(*          Utility procedures used internally by PMPDeque.           *)
(*--------------------------------------------------------------------*)

PROCEDURE DeleteNode   (VAR node : Deques);
(* Deletes specified node from deque                                   *)

VAR temp    : Deques;           (* save pointer for deletion *)

BEGIN
    IF node^.next = node THEN           (* Delete last node in deque *)
       DEALLOCATE (node^.contents, node^.size);
       DEALLOCATE (node, TSIZE (DequeNodes));
       node := NIL;
    ELSE
       temp := node;                    (* Delete specified node     *)
       node := node^.next;
       temp^.previous^.next := node;
       node^.previous := temp^.previous;
       DEALLOCATE (temp^.contents, temp^.size);
       DEALLOCATE (temp, TSIZE (DequeNodes));
    END;
END DeleteNode;

PROCEDURE InsertNodeAtEnd       (node : Deques; VAR deque : Deques);
(* Inserts specified node at end of deque                              *)

BEGIN
    IF Empty (deque) THEN        (* create front *)
       node^.next := node;
       node^.previous := node;
       deque := node;
    ELSE                         (* add to end   *)
       node^.next := deque;
       node^.previous := deque^.previous;
       deque^.previous^.next := node;
       deque^.previous := node;
    END;
END InsertNodeAtEnd;
```

```
(*-------------------------------------------------------------------*)
(*          Procedures exported by PMPDeque.                         *)
(*-------------------------------------------------------------------*)

PROCEDURE CreateDeque           (VAR deque : Deques);
(* Creates the deque by making it NIL                                *)
(* This procedure must be called prior to performing operations on   *)
(* the deque.                                                        *)

BEGIN
    deque := NIL;
END CreateDeque;

PROCEDURE Empty         (deque : Deques) : BOOLEAN;
(* Return TRUE if the deque is NIL                                   *)

BEGIN
    RETURN deque = NIL;
END Empty;

PROCEDURE DestroyDeque (VAR deque : Deques);
(* Destroys the deque by deallocating deque nodes and contents       *)
(* DestroyDeque should be called when the deque is no longer needed  *)
(* in memory.                                                        *)

BEGIN
    WHILE NOT Empty (deque) DO
          DeleteNode (deque);
    END;
END DestroyDeque;

PROCEDURE Full          (size : CARDINAL) : BOOLEAN;
(* Return TRUE if there is no Available memory for another deque node *)

BEGIN
    RETURN (NOT (Available (TSIZE (DequeNodes)) AND Available (size)));
END Full;

PROCEDURE DequeLength (deque : Deques) : CARDINAL;
(* Returns the number of nodes in the deque                          *)

VAR
    current : Deques;           (* cursor used to walk the deque *)
    count   : CARDINAL;         (* node counter *)

BEGIN
    count := 0;

    IF NOT Empty (deque) THEN
       current := deque^.previous;

       REPEAT
             INC (count);
             current := current^.next;
       UNTIL current = deque^.previous;
    END;

    RETURN count;
END DequeLength;

PROCEDURE DequePos      (data : ARRAY OF BYTE; deque : Deques) : CARDINAL;
(* Returns the number of the node containing the data (if found).    *)
(* Returns 0 if not found.  The front of the deque is treated as node *)
(* number one.                                                       *)

VAR
    count,                      (* node counter *)
    pos      : CARDINAL;        (* returned from Lib.Compare *)
    current  : Deques;          (* cursor used to walk the deque *)
```

```
BEGIN
    count := 0;
    IF NOT Empty (deque) THEN
       current := deque^.previous;

       REPEAT
            INC (count);
            current := current^.next;
            pos := Compare (ADR (data), current^.contents, current^.size);
       UNTIL (current = deque^.previous) OR (pos = current^.size);
    END;

    IF pos = current^.size THEN
       RETURN count
    ELSE
       RETURN 0
    END;
END DequePos;

PROCEDURE InDeque       (data : ARRAY OF BYTE; deque : Deques) : BOOLEAN;
(* Returns TRUE if data is found in deque.                                *)

BEGIN
    RETURN (DequePos (data, deque) # 0);
END InDeque;

PROCEDURE Enqueue       (    data : ARRAY OF BYTE;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place data in node at the back of the deque.  Ok will be set to     *)
(* FALSE under the following conditions:  Deque is full.               *)

VAR
    temp        : Deques;       (* used to create new node *)
    dataSize    : CARDINAL;     (* size of data *)

BEGIN
    dataSize := HIGH (data) + 1;
    Ok := NOT Full (dataSize);

    IF Ok THEN
       ALLOCATE (temp, TSIZE (DequeNodes));
       ALLOCATE (temp^.contents, dataSize);
       Move (ADR (data), temp^.contents, dataSize);
       temp^.size := dataSize;
       InsertNodeAtEnd (temp, deque);
    END;
END Enqueue;

PROCEDURE Push          (     data : ARRAY OF BYTE;
                          VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place data in node at the back of the deque.  Ok will be set to     *)
(* FALSE under the following conditions:  Deque is full.               *)

BEGIN
    Enqueue (data, deque, Ok);
END Push;

PROCEDURE Dequeue       (VAR data : ARRAY OF BYTE;
                         VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the front of the deque into data.       *)
(* Remove the front node of the deque.  Ok is set to FALSE upon the    *)
(* following conditions:  deque is empty, size of data does not equal  *)
(* size of contents.                                                   *)

VAR
    dataSize    : CARDINAL;     (* size of data *)
```

```
BEGIN
    Ok := NOT Empty (deque);
    IF Ok THEN
       dataSize := HIGH (data) + 1;

       Ok := deque^.size = dataSize;

       (* size of data must equal the size of contents *)

       IF Ok THEN
          Move (deque^.contents, ADR (data), deque^.size);
          DeleteNode (deque);
       END;
    END;
END Dequeue;

PROCEDURE Pop          (VAR data : ARRAY OF BYTE;
                        VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back of the deque into data.       *)
(* Remove the back node of the deque.  Ok is set to FALSE upon the    *)
(* following conditions:  deque is empty, size of data does not equal *)
(* size of contents.                                                  *)

VAR
  dataSize   : CARDINAL;        (* size of data *)

BEGIN
    Ok := NOT Empty (deque);
    IF Ok THEN
       dataSize := HIGH (data) + 1;

       IF deque^.next = deque THEN          (* only one node *)
          Ok := deque^.size = dataSize;

          (* size of data must equal size of contents *)

          IF Ok THEN
             Move (deque^.contents, ADR (data), deque^.size);
             DeleteNode (deque);
          END;
       ELSE
          Ok := deque^.previous^.size = dataSize;

          (* size of data must equal size of contents *)

          IF Ok THEN
             Move (deque^.previous^.contents, ADR (data), deque^.previous^.size);
             DeleteNode (deque^.previous);
          END;
       END;
    END;
END Pop;

PROCEDURE Serve        (VAR data : ARRAY OF BYTE;  nthItem : CARDINAL;
                        VAR deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from nth node of the deque into data.  Remove the   *)
(* nth node of the deque.  Ok is set to FALSE upon the following      *)
(* conditions:  deque is empty, size of data does not equal size of   *)
(* contents, nthItem is greater than number of nodes in the deque.    *)

VAR
    current        : Deques;     (* used to walk the deque *)
    dataSize,                    (* size of data *)
    numberOfNodes,               (* length of deque *)
    count          : CARDINAL;   (* node counter *)

BEGIN
    Ok := NOT Empty (deque);
```

```
    IF Ok THEN
       numberOfNodes := DequeLength (deque);

       Ok := (nthItem <= numberOfNodes) AND (nthItem > 0);

       (* nthItem must be less than length of deque & greater than 0 *)

       IF Ok THEN
          dataSize := HIGH (data) + 1;

          count := 0;
          current := deque^.previous;

          REPEAT
                INC (count);
                current := current^.next;
          UNTIL count = nthItem;

          Ok := current^.size = dataSize;

          (* size of contents must equal size of data *)

          IF Ok THEN
             Move (current^.contents, ADR (data), current^.size);

             IF count = 1 THEN
                DeleteNode (deque)
             ELSE
                DeleteNode (current)
             END;
          END;
       END;
    END;
END Serve;

PROCEDURE Front         (VAR data : ARRAY OF BYTE;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the front of the deque into data.      *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

VAR
    dataSize : CARDINAL;        (* size of data *)

BEGIN
    Ok := NOT Empty (deque);
    IF Ok THEN
       dataSize := HIGH (data) + 1;
       Ok := deque^.size = dataSize;

       (* size of data must equal size of contents *)

       IF Ok THEN
          Move (deque^.contents, ADR (data), deque^.size);
       END;
    END;
END Front;

PROCEDURE Back          (VAR data : ARRAY OF BYTE;
                             deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back of the deque into data.       *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

VAR
    dataSize : CARDINAL;        (* size of data *)

BEGIN
    Ok := NOT Empty (deque);
```

```
    IF Ok THEN
       dataSize := HIGH (data) + 1;

       Ok := deque^.previous^.size = dataSize;

       (* size of data must equal size of contents *)

       IF Ok THEN
          Move (deque^.previous^.contents, ADR (data), deque^.previous^.size);
       END;
    END;
END Back;

PROCEDURE Top          (VAR data : ARRAY OF BYTE;
                            deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from node at the back (top) of the deque into data. *)
(* Ok is set to FALSE upon the following conditions:  deque is empty, *)
(* size of data does not equal size of contents.                      *)

BEGIN
    Back (data, deque, Ok);
END Top;

PROCEDURE Retrieve     (VAR data : ARRAY OF BYTE;
                            nthItem : CARDINAL;
                            deque : Deques; VAR Ok : BOOLEAN);
(* Place contents from nth node of the deque into data.  Ok is set to *)
(* FALSE upon the following conditions:  deque is empty, size of data *)
(* does not equal size of contents.  nthItem is greater than number   *)
(* of nodes in the deque.                                             *)

VAR
    current        : Deques;     (* used to walk the deque *)
    dataSize,                    (* size of data *)
    numberOfNodes,               (* length of deque *)
    count          : CARDINAL;   (* node counter *)

BEGIN
    Ok := NOT Empty (deque);
    IF Ok THEN
       numberOfNodes := DequeLength (deque);

       Ok := (nthItem <= numberOfNodes) AND (nthItem > 0);

       (* nthItem must be less than length of deque & greater than 0 *)

       IF Ok THEN
          dataSize := HIGH (data) + 1;

          count := 0;
          current := deque^.previous;

          REPEAT
                INC (count);
                current := current^.next;
          UNTIL count = nthItem;

          Ok := current^.size = dataSize;

          (* size of contents must equal size of data *)

          IF Ok THEN
             Move (current^.contents, ADR (data), current^.size);
          END;
       END;
    END;
END Retrieve;

END PMPDeque.
```

## WDS Startup Program

By William D. Smith

Since I've presented most of my units which make up the WDS environment, I will now start with the programs which use these units. This first program is the first program which runs after the operating system is ready. The code file is named "SYSTEM.STARTUP" on the boot disk which causes the operating system to automatically execute it before displaying the main system prompt.

This program starts by adjusting the terminal screen size (`Set_Sc_Size` in `Initialize`) to that contained in the system miscinfo (my terminal has multiple screen sizes and if the system crashes with the terminal set wrong, what you see is not always correct). It then displays a prompt for the user to enter their userid (notice that `NoBreak` is used to disable the system break key). After entering an acceptable one, the main menu is displayed. The three main things the user can do are quit (in my case run off the hard disk or floppy), proceed onward with the startup program supplied by PECAN (which sets up execution off of the ramdisk, more on this later) or have the program prompt for the userid again (which locks out unauthorized users while you are away from your machine). At this time you can also set the date on the boot disk (if it needs changing) and set the system time (if you don't have a real time clock).

PECAN provides a startup program with version IV.2.2 which performs a set of commands (such as zeroing the ramdisk, transfering files from one disk to another, mounting sub-volumes, setting the boot and prefix volumes, specifying the file names of the compiler, editor, etc.) contained in a file called "*LOADFILES.TEXT". I use these two files to setup my system for use.

If you remember from previous NewsLetters, my ramdisk is large enough to contain the whole system, all my utilities, and my complete current project (and still have 2000 blocks free). As any developer knows from experience, when you do testing, the system sometimes crashes and needs re-booting. Except for rare times, the ramdisk is not harmed but if I boot in my normal manner it gets zeroed. Because of this I set up my startup program to be able to boot in many different ways.

How did I do this? First I changed the name of PECAN's startup program from "SYSTEM.STARTUP" to "LOADFILES.CODE". This lets my program execute first (since its named "SYSTEM.STARTUP").

"LOADFILES.CODE" always uses "*LOADFILES.TEXT" as its command file (its imbedded in the program and the user is not given a way of changing it on the fly, use can use patch to change it.). I didn't like this so I developed a process of changing it prior to chaining to it. The function `FixStartup` copies the string specified by the user in the L(oadFiles input line to every place that "*LOADFILES.TEXT" occured in the original code (object code). The locations are place at the end of the last block of the "LOADFILES.CODE" program in the form block number, byte number.

FixStartup locates the last block of the code file (not counting the QuickStart blocks), starts at the end working backward until it reads a block number of zero. — One note, FixStartup does not handle the byte sex problem either when locating the last block or when getting the string locations.

The L(oadFiles input line is limited to 15 characters since that is how much room there is in the string "*LOADFILES.TEXT".

In `Initialize`, the default values for `LoadFile`, `MsgFile`, `StartCode` are blank filled out to their maximum possible length so that the code can be patched with new values without needing re-compiled.

The message file is used to leave yourself a message or when more then one user uses the machine, leave messages for them. Both the load file and message file (or any other text file, just change the message file name) can be read with the eX(amine command. This command uses a function from a unit not covered yet which allows the user to examine a text file (read forward and backward).

```
{ WDS terminal typed I/O unit       [1.05] --- 20 Oct 87 } { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1987 to 1989,  All rights reserved.              }

{ File:         T_Tio_U.Text              Version 1.05    20 Oct 87

  Author:       William D. Smith          Phone: (619) 941-4452
                P.O. Box 1139             CIS: 73007,173
                Vista, CA  92083

  Notice:       The information in this document is the exclusive
                property of William D. Smith.  All rights reserved.
                Copyright (c) 1987 to 1989.

  System:       Power System version IV.2.2

  Compiler:     Power System Pascal Compiler

  Keywords:     WDS T_Tio_U Terminal Typed Input Output Unit

  Description:  WDS terminal typed input / output unit.  This unit contains
                procedures for typed terminal input and output.  The procedures
                are for reading / writing dates, integers, time, and YesNo.

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
20 Oct 87   WDS  1.05  Added Vs_T_Tio_U and its use.
17 Sep 87   WDS  1.04  Added W_Time and G_Time.
31 Aug 87   WDS  1.03  Fixed for version IV.22.
20 Aug 87   WDS  1.02  Made InForm clear lines when input is null.
16 Jul 87   WDS  1.01  Put in version control.
12 Jun 87   WDS  1.00  Derived from F.T_Io.
}
{$I VERSION.TEXT}  { Declares conditional compilation flags }
{$D CSR-}  { Outputs cursor type }
{$D KBD-}  { Outputs keyboard lock status }
unit T_Tio_U;
interface {$ T_Tio_U [1.05] 20 Oct 87 }
uses  Glbs_U;   { WDS globals unit }
const Vc_T_Tio_U   =  4;   { 09 Mar 88 }
      Vs_T_Tio_U   = 'T_Tio_U';
var   Vv_T_Tio_U   : integer;
      DotOk        : boolean;   { Set to false when InForm/W_Dot not wanted }
  procedure W_Int (I, Len, Blank : integer);
  { Write the integer I to the screen.  Use Len spaces.  Leading zeros are
    blank.  Right justified the number.  If I equals Blank, then clear Len
    spaces.  Blank is the null value for the particular type of integer
   }
  procedure W_Date (Date : DateRec);
  { Write Date to the screen in the form "mm-dd-yy".  Uses eight spaces. }
  procedure W_Yn (Yn : YesNo;  Len : integer);
  { Write the value of Yn to the screen.  Use Len spaces. }
  procedure W_Time (Tr : TimeRec);
  { Write the time Tr to the screen in the form "hh:mm".  Uses five spaces. }
  procedure InForm (S1 : Str_63;  I : integer;  S2 : Str_63);
  { Inform.  This procedure writes S1, I, and S2 to the 2nd to last line  of the
    screen.  The cursor is left at the bottom line of the screen for W_Dot.
   }
```

```
  procedure W_Dot (var Count : integer);
  { Write dot.  If Count mod 50 is 0, this procedure clears the current line and
    writes count to the screen.  In all cases this procedure writes a dot at the
    location of the cursor.  Count is incremented by one.  InForm is usually
    called before this procedure to set up the cursor.
  }

{ Typed input procedures.  The cursor is left at the end of the value entered.
  Invalid values cause an error to be displayed and the terminal to beep.  The
  value sent into the function is displayed on the screen as the default value.
  The functions return the last character typed which will be an accept key or
  the abort key.  If an accept key is entered, the value returned is what the
  user entered, otherwise the value is unchanged from the default value.
}

  function G_Int (var Int : integer;  Min, Max, Blank : integer) : char;
  { Get integer.  Min and Max are the minimum and maximum values allowed.  If
    the user enters a blank value (nothing), Int is returned as Blank.  If Blank
    equals Min, the user is not allowed to enter a blank.
  }

  function G_Date (var D : DateRec) : char;
  { Get Date.  Uses eight spaces.  The user inputs a legal date in the form
    "mm?dd?yy", where the "?" can be any non-numeric character.
  }

  function G_Yn (var Yn : YesNo;  Allowed : YnSet) : char;
  { Get yes/no.  Allowed contains the set of allowed values ("", "Yes", "No" or
    "Only") which may be entered by the user.
  }

  function G_Time (var Tr : TimeRec) : char;
  { Get time.  This function allows the user to enter a time in the form
    "hh:mm".
  }

implementation

uses  StrOps_U,     { WDS string conversion ops unit }
      T_Io_U,       { WDS terminal I/O unit }
      OpSys_U;      { WDS to operating system interface unit }

  function I_Len (I : integer) : integer;
  var J : integer;
  begin
    if I < 0 then  begin  J := 1;  I := - I;  end { if }
    else  J := 0;

    if I < 10 then  I := 1
    else if I < 100 then  I := 2
    else if I < 1000 then  I := 3
    else if I < 10000 then  I := 4
    else  I := 5;

    I_Len := I + J;
  end { I_Len };

  procedure W_Int { (I, Len, Blank : integer) };
  var S : Str_81;
  begin
    if (Len = 0) and (I <> Blank) then  Len := I_Len (I);

    S [0] := chr (Len);
    fillchar (S [1], Len, ' ');

    if I <> Blank then  I_into_S (I, S, 1, Len);

    W_Str (S);
  end { W_Int };
```

```
procedure W_Date { (Date : DateRec) };
var S : Str_9;
begin
  S [0] := chr (8);
  fillchar (S [1], 8, ' ');
  D_into_S (Date, S, 1);
  W_Str (S);
end { W_Date };

procedure W_Yn { (Yn : YesNo;  Len : integer) };
var Ss : Str_5;  S : Str_81;
begin
  S [0] := chr (Len);
  fillchar (S [1], Len, ' ');
  Yn_to_S (Yn, Ss);
  if length (Ss) > 0 then
    moveleft (Ss [1], S [Len - length (Ss) + 1], length (Ss));
  W_Str (S);
end { W_Yn };

procedure W_Time { (Tr : TimeRec) };
var S : Str_5;
begin
  if Tr = NullTad .T then
    S := '     '
  else
    begin
      S := '00:00';
      T_into_S (Tr, false, S, 1);
    end { else };
  W_Str (S);
end { W_Time };

procedure InForm { (S1 : Str_63;  I : integer;  S2 : Str_63) };
var LastLine, Xx : integer;
begin
  if DotOk then
    begin
      LastLine := Height - 1;
      if Y = LastLine then  Xx := X
      else  Xx := 0;
      Set_Xy (0, LastLine - 1);
      if (length (S1) = 0) and (I = Null) and (length (S2) = 0) then
        begin
          C_Eol;
          SetXy (0, Y + 1);
          C_Eol;
        end { if }
      else
        begin
          W_Str (S1);
          W_Int (I, 0, Null);
          W_Str (S2);
          C_Eol;
        end { else };
      Set_Xy (Xx, LastLine);
      S_Info ('');
    end { if };
end { InForm };
```

```
procedure W_Dot { (var Count : integer) };
var Xx : integer;  S : Str_9;
begin
  if DotOk then
    begin
      if Count mod 50 = 0 then
        begin
          if X > 56 then  Xx := X - 56
          else  Xx := 0;
          S := '<   0>';
          I_into_S (Count, S, 2, 4);
          Set_Xy (Xx, Y);  W_Str (S);  C_Eol;
          S_Info ('');
        end { if };
      W_Char ('.');
      Count := Count + 1;
    end { if };
end { W_Dot };

function G_Int { (var Int : integer;  Min, Max, Blank : integer) : char };
var Ch    : char;
    Xx    : integer;
    L     : integer;  { length of Int }
    Len   : integer;  { length of string }
    Tmp   : integer;
    Blnk  : integer;
    Done  : boolean;
    S     : Str_31;
begin
  Xx := X;
  L := I_Len (Int);
  Len := I_Len (Max);
  Tmp := I_Len (Min);
  if Tmp > Len then  Len := Tmp;
  if Min = Blank then  Blnk := Blank - 1
  else  Blnk := Blank;
  Done := false;
  repeat
    if Int = Blnk then  S := ''
    else
      begin
        S [0] := chr (L);
        fillchar (S [1], length (S), ' ');
        I_into_S (Int, S, 1, L);
      end { else };
    Ch := G_Str (S, Len);
    if Ch = chr (Esc) then  Done := true
    else
      begin
        Crunch_Str (S, S);
        if length (S) = 0 then
          if Min = Blank then
            Error ('Value required')
          else
            begin  Int := Blank;  Done := true;  end { else }
        else if S_to_I (S, Tmp, Min, Max) then
          begin  Int := Tmp;  Done := true;  end { else if }
```

```
        else
          begin  {12345678 1 2345678 2 2345}
            S := 'Invalid, (      to      )';
            I_into_S (Min, S, 11, 5);
            I_into_S (Max, S, 20, 5);
            Error (S);
          end { else };
      end { else };
    SetXy (Xx, Y);
  until Done;
  W_Int (Int, Len, Blnk);
  G_Int := Ch;
end { G_Int };

function G_Date { (var D : DateRec) : char };
var Xx    : integer;
    Tmp   : DateRec;
    Ch    : char;
    Done  : boolean;
    S     : Str_9;
    Ss    : Str_9;
begin
  Xx := X;
  if D = NullTad .D then  Ss := ''
  else
    begin  {12345678}
      Ss := '        ';
      D_into_S (D, Ss, 1);
    end { else };
  Done := false;
  repeat
    S := Ss;
    Tmp := D;
    Ch := G_Str (S, 8);
    if Ch = chr (Esc) then  Done := true
    else if S_to_D (S, Tmp) then
      begin
        D := Tmp;
        Done := true;
      end { else if }
    else  Error ('Invalid, (check illegal dates)');
    SetXy (Xx, Y);
  until Done;
  W_Date (D);
  G_Date := Ch;
end { G_Date };

function G_Yn { (var Yn : YesNo;  Allowed : YnSet) : char };
var Xx    : integer;
    Len   : integer;
    Ch    : char;
    Done  : boolean;
    Tmp   : YesNo;
    S     : Str_31;
begin  { G_Yn }
  Xx := X;
  Done := false;
  if Only in Allowed then  Len := 4
  else  Len := 3;
```

```
    repeat
      Yn_to_S (Yn, S);
      Ch := G_Str (S, Len);
      if Ch = chr (Esc) then  Done := true
      else if S_to_Yn (S, Tmp, Allowed) then
        begin  Yn := Tmp;  Done := true;  end { else if }
      else
        begin
          S := 'Invalid, (Yes, No)';
          if Only in Allowed then
            insert (', Only', S, pos (')', S));
          Error (S);
        end { else };
      SetXy (Xx, Y);
    until Done;
    W_Yn (Yn, Len);
    G_Yn := Ch;
  end { G_Yn };

  function G_Time { (var Tr : TimeRec) : char };
  var Xx    : integer;
      Ch    : char;
      Done  : boolean;
      Tmp   : TimeRec;
      S     : Str_5;
      Ss    : Str_5;
  begin
    Xx := X;  Done := false;
    if Tr = NullTad .T then  Ss := ''
    else
      begin
        Ss := '00:00';
        T_into_S (Tr, false, Ss, 1);
      end { else };
    repeat
      S := Ss;
      Ch := G_Str (S, 5);
      if Ch = chr (Esc) then  Done := true
      else if length (S) = 0 then
        begin  Tr := NullTad .T;  Done := true;  end { else if }
      else if S_to_T (S, Tmp) then
        begin  Tr := Tmp;  Done := true;  end { else if }
      else  Error ('Invalid time');
      SetXy (Xx, Y);
    until Done;
    W_Time (Tr);
    G_Time := Ch;
  end { G_Time };

begin
  Vv_T_Tio_U := Vc_T_Tio_U;
  Ck_Version (Vv_Glbs_U, Vc_Glbs_U, Vs_T_Tio_U, Vs_Glbs_U);
  Ck_Version (Vv_StrOps_U, Vc_StrOps_U, Vs_T_Tio_U, Vs_StrOps_U);
  Ck_Version (Vv_T_Io_U, Vc_T_Io_U, Vs_T_Tio_U, Vs_T_Io_U);
  DotOk := true;
  *** ;
end {$Q- T Tio U }.
```

```
{ System startup (Wyse IV)          [2.09] --- 09 Mar 88 } { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1985 to 1989,  All rights reserved.           }

{ File:          Startup.Text              Version 2.09    09 Mar 88

  Author:        William D. Smith          Phone: (619) 941-4452
                 P.O. Box 1139             CIS: 73007,173
                 Vista, CA  92083

  Notice:        The information in this document is the exclusive
                 property of William D. Smith.  All rights reserved.
                 Copyright (c) 1985 to 1989.

  System:        Power System version IV.2.2

  Compiler:      Power System Pascal Compiler

  Keywords:      WDS System Startup Program

  Description:   Startup program.  This is the first program executed when the
                 system boots up.

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
09 Mar 88   WDS  2.09  Changed to use F_Io_U.
07 Feb 88   WDS  2.08  Used Get_MyHelp.
06 Feb 88   WDS  2.07  Changed to reflect changes in T_Io_U for C_Sc.
24 Jan 88   WDS  2.07  Added eX(amine command.
20 Oct 87   WDS  2.06  Changed to use Vs_ names.
28 Sep 87   WDS  2.05  Added H(elp command.
23 Sep 87   WDS  2.04  Added C(heckLoadFile command.
16 Sep 87   WDS  2.04  Fixed some errors in G_Time.
14 Sep 87   WDS  2.03  Fixed so that the name of LoadFiles.Text can be changed.
14 Sep 87   WDS  2.02  Changed to use Display_U.
20 Aug 87   WDS  2.01  Added Caps to AppendText.
19 Jun 87   WDS  2.00  Used WDS units.
04 Apr 87   WDS  1.11  For IV21, chain to *SYSDATE before creating ramdisk.
06 Nov 86   WDS  1.10  Changed the way volumes are set.
31 Oct 86   WDS  1.09  Added code to use LoadFuncKeys unit.
21 Jul 86   WDS  1.08  Changed ".FCN" to ".FKYS".
14 Jun 86   WDS  1.07  Added use of Version.Text.
17 Feb 86   WDS  1.06  Added stuff for IV.21 startup.
29 Nov 85   WDS  1.05  Changed prefix vol to FDB1TXT:.
12 Sep 85   WDS  1.04  Added conditional compilation for setting prefix.
07 Sep 85   WDS  1.03  Added writing out prefix volume.
07 Sep 85   WDS  1.02  Added AOS conditional compilation.
31 Aug 85   WDS  1.01  Added command to chain to set date program.
05 Feb 85   WDS  1.00  Added function key definitions.
}
{$I VERSION.TEXT}  { Declares conditional compilation flags }
program Startup;
uses  Glbs_U;              { WDS globals unit }
      StrOps_U,            { WDS string ops unit }
      F_Io_U,              { WDS file I/O unit }
      OpSys_U,             { WDS operating system interface unit }
      T_Io_U,              { WDS terminal I/O unit }
      T_Tio_U,             { WDS terminal typed I/O unit }
      Display_U,           { WDS display unit }
      Wild,
      Dir_Info,
      Command_Io;
```

```
const Version   = '[2.09]';{09 Mar 88}
      MaxCount  = 4;
      Other     = 'FFFF';
      Other_Ok  = false;

var   Ch        : char;
      Cx        : integer;
      Dy        : integer;
      Ly        : integer;
      My        : integer;
      Px        : integer;
      Break_Ok  : boolean;
      Done      : boolean;
      N_Menu    : boolean;
      LoadFile  : Str_23;
      MsgFile   : Str_23;
      HelpFile  : Str_23;
      StartCode : Str_23;
      Cmds      : CharSet;

  procedure S_Tad (Tad : TadRec);
  { Show time and date }
  var S : Str_31;
  begin {12345678 1 2345678 2 234}
    S := '00-00-00  00:00  00:00am';
    D_into_S (Tad .D, S, 1);
    T_into_S (Tad .T, false, S, 11);
    T_into_S (Tad .T, true,  S, 18);
    W_Str (S);
  end { S_Tad };

  procedure SetTime;
  var Tad, SaveTad : TadRec;
      S : Str_63;
  begin
    SetXy (Cx + 10, Dy);
    Get_Sys_Tad (Tad);
    SaveTad := Tad;
    if G_Time (Tad .T) <>
                    chr (Esc) then
      if Tad <> SaveTad then
        begin
          Tad .D := NullTad .D;
          Set_Sys_Tad (Tad);
          S_Time;
        end { if if };
    SetXy (Cx, Dy);
    Get_Sys_Tad (Tad);
    S_Tad (Tad);
  end { SetTime };

  procedure SetDate;
  var Tad, SaveTad : TadRec;
  begin
    SetXy (Cx, Dy);
    Get_Sys_Tad (Tad);
    Tad .T := NullTad .T;
    SaveTad := Tad;
  { Get Date from user }
    if G_Date (Tad .D) <>
                    chr (Esc) then
      if Tad .D <> NullTad .D then
        if Tad <> SaveTad then
          begin
            Set_Sys_Tad (Tad);
            if D_ChangeDate ('*',
        Tad .D, [D_Vol]) <> D_Okay then
              Error ('Error ...
  changing date on system volume "*"');
          end { if if if };
    SetXy (Cx, Dy);
    Get_Sys_Tad (Tad);
    S_Tad (Tad);
  end { SetDate };

  function G_Id : boolean;
  var Done, NotDone : boolean;
      I : integer;  Id : Str_31;
  begin
    G_Id := false;
    Done := false;
    C_Sc (true);
    N_Menu := true;
    repeat
      NotDone := true;
      SetXy (0, Y + 1);
      W_Str ('User id -> ');
      Beep;
      Id := '';  I := 0;
      while NotDone do begin
        Ch := G_Chars ([' '..'~'],
                        true);
        if Ch = chr (Esc) then
          begin
            NotDone := false;
            Done := true;
          end { if }
        else if Ck_Accept (Ch) then
          begin
            if ValidUser (Id) then
              begin
                NotDone := false;
                Done := true;
                G_Id := true;
              end { else if }
            else  NotDone := false;
          end { else if }
        else if I = 31 then
          NotDone := false;
        else
          begin
            I := I + 1;
            Id [0] := chr (I);
            Id [I] := Ch;
            W_Char ('#');
          end { else };
      end { while };
    until Done;
  end { G_Id };
```

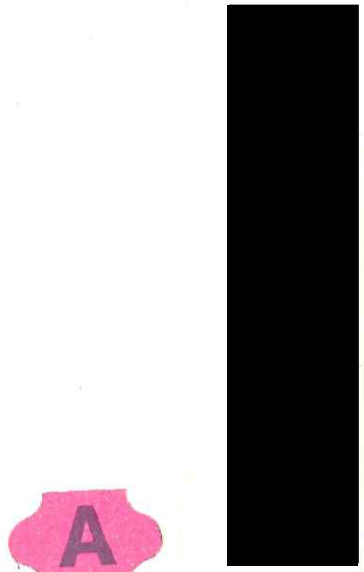
```
function FixStartup (CodeName : Str_23;  LoadFile : Str_23) : boolean;
type BlockByte = record
                   BlockNum : integer;
                   ByteNum  : integer;
                 end { BlockByte };
var  I          : integer;
     Msg        : integer;
     LastBlock  : integer;
     F          : FibPtr;
     CkBlock    : array [0..127] of BlockByte;
     Block      : packed array [0..511] of Byte;

  procedure SetMsg (var Msg : integer;  Ior : integer);
  begin
    if Ior = M_NoError then  Msg := M_Unknown
    else  Msg := Ior;
  end { SetMsg };

begin { FixStartup }
  Msg := M_NoError;
  if OpenFile (F, CodeName, BlkFile, true, Msg) then
    begin
      if BlockIo (F, CkBlock, 1, 0, true) <> 1 then  SetMsg (Msg, ioresult)
      else
        begin
          if CkBlock [106] .BlockNum > 0 then
            LastBlock := CkBlock [106] .BlockNum - 1
          else  LastBlock := File_Size (F) - 1;
          if BlockIo (F, CkBlock, 1, LastBlock, true) <> 1 then
            SetMsg (Msg, ioresult)
          else
            begin
              I := 127;
              while (Msg = M_NoError) and
                    (CkBlock [I] .ByteNum > 0) do begin
                if BlockIo (F, Block, 1,
                            CkBlock [I] .BlockNum, true) <> 1 then
                  SetMsg (Msg, ioresult)
                else
                  begin
                    moveleft (LoadFile [0], Block [CkBlock [I] .ByteNum],
                              length (LoadFile) + 1);
                    if BlockIo (F, Block, 1,
                                CkBlock [I] .BlockNum, false) = 1 then
                      I := I - 1
                    else  SetMsg (Msg, ioresult);
                  end { else };
              end { while };
            end { else };
        end { else };
      CloseFile (F, false);
    end { else };
    if Msg = M_NoError then  FixStartup := true
    else  begin  Err_Msg (Msg, CodeName);  FixStartup := false;  end;
  end { FixStartup };

procedure S_Menu (Prompt : boolean);
var Tad : TadRec;
begin
  if Prompt then
```

```
    begin
      C_Sc (false);
      Px := S_Prompt ('Startup: H(elp G(oLoadFile U(serId eX(amine Q(uit',
                      Version, false);
    end { if };
  SetXy (2, 5);        W_Str ('D(ate & T(ime ------> ');  Dy := Y;
  Cx := X;
  Get_Sys_Tad (Tad);   S_Tad (Tad);
  SetXy (2, Y + 2);    W_Str ('L(oadFile ----------> ');  Ly := Y;
  W_Str (LoadFile);
  SetXy (2, Y + 2);    W_Str ('M(essage file ------> ');  My := Y;
  W_Str (MsgFile);
  S_Info ('');  N_Menu := false;
end { S_Menu };

procedure Examine;
var Ch : char;  Xx : integer;
begin
  Py := Py + 1;
  Xx := S_Prompt ('Examine: L(oadFile M(essageFile Q(uit', Version, false);
  repeat
    SetXy (Xx, Py);
    Ch := G_Char (['L', 'M', 'Q'], false, true);
    case Ch of
      'L' : if DisplayFile (Py + 1, Null, LoadFile) then  N_Menu := true;
      'M' : if DisplayFile (Py + 1, Null, MsgFile) then  N_Menu := true;
    end { cases };
    if N_Menu then  S_Menu (false);
  until Ch = 'Q';
  SetXy (0, Py);  C_Eol;  Py := Py - 1;
end { Examine };

procedure Initialize;
var S : Str_15;
begin
  S := 'Startup';
  Ck_Version (Vv_Glbs_U, Vc_Glbs_U, S, Vs_Glbs_U);
  Ck_Version (Vv_StrOps_U, Vc_StrOps_U, S, Vs_StrOps_U);
  Ck_Version (Vv_F_Io_U, Vc_F_Io_U, S, Vs_F_Io_U);
  Ck_Version (Vv_OpSys_U, Vc_OpSys_U, S, Vs_OpSys_U);
  Ck_Version (Vv_T_Io_U, Vc_T_Io_U, S, Vs_T_Io_U);
  Ck_Version (Vv_T_Tio_U, Vc_T_Tio_U, S, Vs_T_Tio_U);
  Ck_Version (Vv_Display_U, Vc_Display_U, S, Vs_Display_U);
  Set_Sc_Size (AdjustSc, AdjustSc);
  Break_Ok := not NoBreak (Show);
  if NoBreak (On) then ;
  repeat  until G_Id (false);
  if Break_Ok then
    if NoBreak (Off) then ;
  LoadFile  := 'ZBOOT.TEXT     ';
  MsgFile   := 'MSG.TEXT              ';
  StartCode := 'LOADFILES.CODE        ';
  Crunch_Str (LoadFile, LoadFile);
  Crunch_Str (MsgFile, MsgFile);
  Crunch_Str (StartCode, StartCode);
  Get_MyHelp (S, HelpFile);
```

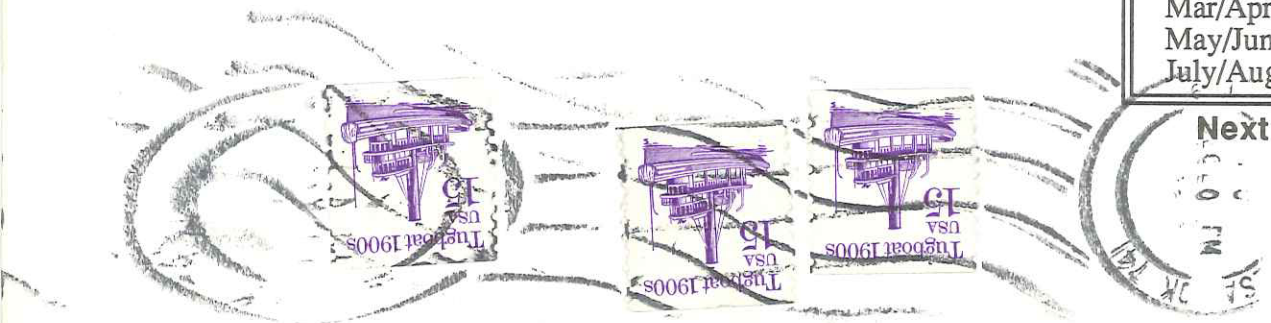
```
    Done := false;  N_Menu := true;
    Cmds := ['D', 'G', 'H', 'L', 'M', 'Q', 'T', 'U', 'X'];
    S_Msg (false, false, CopyRight);
  end { Initialize };

begin { Startup }
  Initialize;
  repeat
    if N_Menu then  S_Menu (true);
    SetXy (Px, Py);
    case G_Char (Cmds, false, true) of
      'D' : SetDate;
      'G' : begin
              Cursor (Off);
              if FixStartup (StartCode, LoadFile) then
                begin
                  S_Msg (false, false, 'Dismounting all subsidiary volumes');
                  if D_DisMount ('=') = D_Okay then ;
                  Chain (concat (StartCode, '.'));   { Performs loadfiles }
                  Done := true;  C_Msg (1);
                end { if };
              Cursor (Pop);
            end { case 'C' };
      'H' : if DisplayFile (My + 1, Null, HelpFile) then ;
      'L' : begin
              SetXy (Cx, Ly);
              if G_Str (LoadFile, 15) <> chr (Esc) then
                begin
                  CapStr (LoadFile);
                  SetXy (Cx, Ly);  W_Str (LoadFile);
                  C_Eol;
                end { if };
            end { case 'M' };
      'M' : begin
              SetXy (Cx, My);
              if G_Str (MsgFile, 23) <> chr (Esc) then
                begin
                  AppendText (MsgFile, true);
                  SetXy (Cx, My);  W_Str (MsgFile);  C_Eol;
                end { if };
            end { case 'M' };
      'Q' : Done := true;
      'T' : SetTime;
      'U' : begin
              if NoBreak (On) then ;
              repeat  until G_Id (false);
              if Break_Ok then
                if NoBreak (Off) then ;
            end { case 'U' };
      'X' : Examine;
    end { cases };
  until Done;
  C_Msg (0);
end {$Q- Startup }.
```

USUS
P.O BOX 1148
LA JOLLA, CA 92038

ADDRESS CORRECTION REQUESTED
FIRST CLASS MAIL







Nov/Dec 1989

# NewsLetter



Volume 3
William D. Smith, Editor Number 8



NewsLetter Publication Dates

| NewsLetter | Due date Code/Forms | Due date Articles | Due date Short stuff |
|---|---|---|---|
| Jan/Feb 90 | 01/01/90 | 01/08/90 | 01/15/90 |
| Mar/Apr 90 | 03/05/90 | 03/12/90 | 03/19/90 |
| May/Jun 90 | 05/07/90 | 05/14/90 | 05/21/90 |
| July/Aug 90 | 07/09/90 | 05/16/90 | 05//23/90 |

**Next NewsLetter coming? UNKNOWN.**