March 1989

USUS NewsLetter

All the News that Fits, We Print

Volume 3

William D. Smith, Editor

Number 3

From the Editor

by William D. Smith

Welcome to our new officers and staff. USUS now has a new President, **Alex Kleider**, a Secretary, **Howard J. Sweet** and a publisher **Robert Geeslin**. Robert will handle printing and mailing the NewsLetter, taking some work off of Hays' hands.

The middle section of this NewsLetter is the current USUS directory as of March 1st, 1989. It is setup so that it can be removed and used separately.

Thanks Alex for submitting one of his modules for inclusion in this NewsLetter.

This NewsLetter is a little bigger then normal, partly because of the directory and partly from the postage cost. For first class mail, which is what we are sending out the NewsLetter as, it cost 25¢ for the first half ounce and 20¢ for each additional half ounce. Twelve pages make up a half ounce, so I to make the NewsLetter end up a multiple of twelve pages, usually 24. With the directory and the stuff Alex submitted, I was over 24 pages so aimed for 36. This did not leave enough room for one of my units I was going to put in (string ops), so it will appear in the next NewsLetter.

Q & / | A? ...

***Robert Spitzer** (CIS: 75226,3643) posted the following question to MUSUS:*

Anyone have any suggestions about the following problem? I will post my solution (kluge) later; I am sure one of the gurus here has come up with a neater one, though. It is the case of the mysterious vanishing GOTO

Consider initializing a large, complex, multi-tasking program (for example, a 43,000 line operating system). You need to go through a very careful sequence of initialization. This could fail at any step. If it does, you want to backtrack exactly through a de-initialization sequence. You do not want to deinitialize stuff that wasn't initialized (like semaphores, tasks, etc) because this can blow up. Dr. Wirth, in a massive triumph of "purity of thought" over practicality, has eliminated the GOTO from Modula. The "structured coding" disaster would look like this:

```
init (a);
IF fail to init (b), DE-init (a);
IF fail to init (c), DE-init (b), DE-init (a);
IF fail to init (d), DE-init (c), DE-init (b),
                     DE-init (a);
(notice the endless duplication!)
```

and so on (yes, one does outgrow structured coding!). Short of kluging a "computed GOTO" (remember FORTRAN?), I have come up with a crude way to simulate it; but I am wondering if there is any elegant solution, short of abort/hit reset switch?

I suggested the following (the iterative method):

How about: (rough pseudo code)

```
I := FirstInit;

repeat
  case I of
    FirstInit : Ok := Init (a);
    FirstInit + 1 : Ok := Init (b);
     ...
    LastInit : Ok := Init (z);
  end { cases };
  if Ok then  I := I + 1;
until not ok or (I = LastInit);

if not Ok then
  for I := I down to FirstInit do begin
    case I of
      FirstInit : De-init (a);
      FirstInit + 1 : De-init (b);
       ...
      LastInit : De-init (z);
    end { cases };
  end { for };
```

Jim Merritt (CIS: 76703,3030) responded with:

William Smith has already given you an iterative solution to the problem if you can make the assumption that failure to initialize module n requires the backwards de-initialization of modules 0..n-1 without any gaps in the sequence. Sometimes, of course, you can plod along even if some of your modules do not initialize properly (i.e., graceful system degradation). You will still have to deinitialize the ones that were OK in backward order. I like to put de-initialization "orders" on a stack (usually implemented as a linked list), which I unwind whenever the time for deinitting comes. If you don't like to do it with a stack, then you could perhaps keep track of the successful; inits with a bitset (or a larger set, if your implementation permits), and in a de-init loop similar to Bill's, check to see if any particular module is in the set of initialized-module set. When you get a "hit," you call the appropriate de-init routine. The latter kind of strategy works well for me in both Pascal and Modula. I am curious to learn how you solved your problem.

***Tom Cattrall** (CIS:72767,622) also answered:*

Bob-- It's possible to do that in a clean manner without goto's. First, allocate an array or list of elements containing procedure variables. Then whenever code anyplace does something that it later wants to have undone, it calls the terminate module with a procedure variable to add to the list. At termination your code calls the termination module which walks the list in the proper order making the various calls.

I prefer the procedure variable list to a set or some other flag mechanism because it is open ended and the termination module needn't change as modules are added or deleted from the system.

On any unexpected terminations you insert a call to the DoAllTerminations procedure besides the call you'd have on normal exit (if any) from the program. Another extension is to pass a priority along with the procedure variable. This would affect the order of the terminations.

The top level of your system then looks like:

```
IF DoAllInits THEN
  GetToWork;
END;
DoAllTermination;
  (* call termination module *)
```

The function DoAllInits looks like:

```
IF NOT InitA THEN
  RETURN FALSE;
END;

IF NOT InitB THEN
  RETURN FALSE;
END;

  ...

IF NOT InitX THEN
  RETURN FALSE;
END;

RETURN TRUE;
```

The various Init* procedures do the required initialization and put any needed terminate procedure calls into the terminate list.

Administrator Says

by Hays Busch

I hope all of you find the USUS Member Roster interesting and helpful. (The last full listing was published in January, 1982, so it's been a long time since there's been a new one!) Now you can find out if there are any other USUS members in your town, county, state or country. And in some areas, you may well be the only member! But, that very fact gives each of you a chance to help USUS get back some of the members it has lost over the years. Here are a few ways you can do this.

1) For those of you who have been members for some time, scan the list for names of members you recall from the past. If you don't find them listed and know how to get in touch with them, give them a call or drop them a note telling them what USUS is doing now. If you don't know how to reach them, chances are pretty good I have them on one of our "Expired Member Lists". Give me their last name for sure (first name if you can...and city or state) and I MAY be able to get you their last know address and telephone number. My experience is that a personal contact is the best way to get back a member who has dropped out but is st[illegible] interested in UCSD Pascal.

2) We tried the idea of having a "State Delegate" for USUS about a year ago. I have no idea if it worked, since the fellow who volunteered to do it, flunked out on me. He either did not get any results, or never gave them to me if he did! The concept was that the State Delegate would write all expired members in a state to see if they would be interested in becoming USUS members again. Since I have all the stuff from the last try, we sure could give it another "shot". This has to be done on a "one state at a time" basis. If we get results from the first try, we can on to another state. All I need to get this started is a note from you saying you'd be willing to become a "State Delegate". Earliest postmark gets the first crack at it. How about it?

3) If you are planning to sell some stuff at a local computer swap, USUS might be able to gain some members if you would set up a demonstration of the System and the Language. Several USUS members have done this in the past and a couple are planning to do such a swap in Washington in the next few months. Let us know at least a month in advance, and we can get stuff to you that can help. This could include a demo of PowerTools and programs from the SWLibe. Each situation varies, so lets work out the details.

4) If you belong to a local computer club, some of the other members may be interested in Pascal or Modula2. Ask around. If you get enough who want to join, a "user group discounted membership fee" can be approved by the USUS Board. In the past, this activity has worked quite well for the TI 99/4a group.

USUS still needs one member to take charge of New Membership Development. If that's something you would like to take on, USUS can sure use your help for this kind of activity. With Board of Directors approval, we can spend small amounts of money for this kind of activity. The real test is getting enough new members from the effort to cover the cost of the effort. If that happens, the activity is worthwhile!

USUS now has a new "Publisher" for the NewsLetter. His name is Bob Geeslin. He took over the responsibility for printing and mailing this issue and will continue to do this for some time to come. Saves me about a day per month and I appreciate his help very much.

A note about mail turn-around time. It takes about two weeks for me to get mail you send to the La Jolla P.O. Box. This is because William Smith, our NL Editor picks it up about once a week and forwards it to me in Colo. So depending on the workload, I may not get back to you for 4 to 6 weeks from the time you mail something to me. (I try to stay within 4-weeks, but its not always possible.) So, if something is in a "hurry" you can always write me at my home address (see Roster listing) or call information for my telephone number. I'm home most days and evenings.

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

Bank Balance	$7,183.79	12-31-89
Income - January 1989		
Dues:		(new/renew)
Student	0.00	0/0
General	785.00	2/19
Professional	100.00	0/1
Institutional	0.00	0/0
Other Income:		
Library fees	7.00	
CIS	44.84	
Total Income:	936.84	
Expenses - January 1989		
Administrator:		
CIS	27.27	
Telephone	6.25	
Photocopies	1.35	
Postage	234.07	
Printing	320.14	
Supplies	12.05	
Other:		
Mail from La Jolla	5.20	
Bank charge	1.00	
Total Expenses	$607.33	
Bank Balance	$7,513.30	01-31-89

Board of Directors Minutes (Jan. 17, 1989)
by Samuel B. Bassett

MINUTES OF THE SPECIAL MEETING OF THE BOARD OF DIRECTORS OF USUS, INC., HELD IN ROOM 1 OF THE MUSUS FORUM TELE-CONFERENCING FACILITY ON THE COMPU-SERVE INFORMATION SERVICE, JANUARY 17, 1989.

Present at the meeting were:

User ID	Name
76314,1364	Sam'l Bassett, Board Chaircritter
73447,2754	Henry Baumgarten, Board Member
72401,1417	Frank Lawyer, Board Member
70260,306	A. Hays Busch, Administrator
72747,3126	Robert Clark, Treasurer
73007,173	William Smith, Assistant SysOp, NewsLetter Editor
72135,1667	Harry Baya, Member
71515,447	Alex Kleider, Member

Matters dealt with were:

Ratification of Minutes

The Chaircritter moved that the minutes of on-line Board meetings up to and including the December 1988 meeting, as published in Data Library 1 & 8 of MUSUS and in the NewsLetter, be accepted as official.

The motion passed by a unanimous vote of the Officers & Directors present.

Budget

The Board discussed the proposed Budget, predicated on 250 members, posted in Section 8 of MUSUS by Bob Clark. Frank was concerned that USUS would have enough money to publish 9 NewsLetters, and that there were no big expenses hiding in the bushes. He was assured that there was, probably wasn't, and the Budget was accepted by vote of all present.

The President Problem

The Chaircritter announced that he had not been able to contact Weber Baker, and that USUS was thus officially without a President, as per the Resolution adopted at the December meeting. Bob Clark mentioned that Alex Kleider had said he would be willing to serve if asked, Frank Lawyer asked why he had to be asked -- wasn't he willing to volunteer? Henry Baumgarten formally nominated Alex, and Bob Clark seconded the nomination. The Chaircritter proposed the following motion:

"The Board of Directors of USUS, Inc. hereby asks Alex Kleider, who had indicated his willingness to do so, to accept the duties of President of USUS, Inc."

The motion was carried unanimously

The Secretary Problem

USUS is also without a Secretary, and Harry & Frank had volunteered to get in touch with Howard Sweet, and find out if he were willing to accept the post (or the duties, for that matter). When asked by the Chaircritter whether they had done so yet, Frank said he had tried calling, but had gotten no answer. He said that he would continue trying. After some more general chat, the Chaircritter ruled the matter "still pending" and moved on to:

Harry's Statement of Policy on Opening Up MUSUS / USUS *(following these minutes)*

. . . which had been posted in Section 8. The Chaircritter opined that he rather liked it, and that when asked earlier, Henry had said that his only worry was that USUS might lose exclusively p-System types. Harry said he would like to post the motion to Section 1 to invite comment by other members, and Henry opined that it should also go into the next newsletter. The Chaircritter agreed, and tried to ask William to take care of both matters, but found that he had dropped out of the conference *(my CompuServe node went down)*. Frank said that since USUS and MUSUS are now effectively open, why bother with being so formal? The Chaircritter replied that since USUS has had the reputation of being a tight, closed little group, he wanted to do something formal to change that. At this point, the motion was tabled, pending further comments from the membership.

Alex is asked

At about this time (11:15 EST) Alex Kleider joined the conference, and was brought up to date about being asked to be President. He replied modestly that he probably ought to chat with the

Directors directly before giving an answer. Frank said he thought Alex should just accept forthwith, and before he and Alex could get into it, the Chaircritter told the two of them to go in a corner and discuss it off-line, clearing the board for discussion of:

Bob Spitzer's motion in regard to JPAM

In a series of messages on MUSUS, Bob Spitzer had argued that USUS should approach the publisher(s) of the Journal of Pascal, Ada, and Modula-2 (JPAM), with the purpose of making JPAM the "Official Journal of USUS". the two chief advantages of this idea are: 1) No expense or hassle of editing, printing, and mailing Journals; and 2) Exposure of USUS to a large number of users of other Pascals and Modula-2s. The Chaircritter put it that the issue was whether to go ahead with the approach to JPAM to see where it leads, and to find someone to do the approaching.

A long discussion ensued, in which no one expressed a great deal of either enthusiasm or antipathy for the idea, and it turned out that Henry was willing to talk to Prof. Weiner, the Editor, and try to contact the Publisher. the Chaircritter distilled the discussion into a motion:

"That the Board of Directors of USUS, Inc., feeling that an association with JPAM might be in the best interests of both parties, authorizes discussions to begin which may lead to a wider and more concrete definition of that association."

This wording was accepted by all present.

Having run out of Agenda items and steam, the Board resolved to adjourn until 7 PM PST / 8 PM MST / 9 PM CST / 10 PM EST February 7th, 1989, and did.

The Special Meeting of the Board on Compuserve was adjourned at 12:00 midnight on January 17/18th, 1989.

Minutes submitted by:
Samuel B. Bassett

Harry's statement as posted to MUSUS and refered to in the preceding minutes.

Fm: Harry Baya, 72135,1667
10-Jan-89 07:44:13

To: Usus Members
Sb: Broaden USUS/MUSUS

This is a first draft of a motion reflecting USUS's intent to broaden its scope to include a wider range of programming languages and environments than it has in the past. I found it difficult to be very specific about this and have tried to use the broadest possible wording.

Up until this date the primary stated focus of USUS, and its CompuServe Forum, MUSUS, has been UCSD Pascal and the UCSD micro-computer development environment. It is now our desire to broaden our focus to: (a) reflect our interests and activities related to the Modula-2 language and (b) include other programming languages in the Pascal/Modula-2 family.

Toward that end (a) we note that USUS has in the past provided resources and support to the community of users of UCSD Pascal and the UCSD micro-computer development environment and (b) we now state that USUS will now provide those same resources and support to the community of users of all variants of Pascal, Modula-2 and closely related languages. Being aware that this wording is very broad, USUS retains the right to include or exclude under its focus umbrella any particular computer language or related activity that it chooses.

We state further that it is not our intent to usurp or replace the activities of other existing user groups or CompuServe Forums, but rather to meet needs not met elsewhere and to support the needs and interests of a community which includes both the individual users of included languages and other related user groups.

Board of Directors Minutes (Jan. 17, 1989)
by Samuel B. Bassett

MINUTES OF THE SPECIAL MEETING OF THE BOARD OF DIRECTORS OF USUS, INC., HELD IN ROOM 1 OF THE MUSUS FORUM TELE-CONFERENCING FACILITY ON THE COMPU-SERVE INFORMATION SERVICE, BEGINNING AT 10:09 PM EST FEBRUARY 7TH, 1989.

Present at the meeting were:

User ID	Name
76314,1364	Sam'l Bassett, Board Chaircritter
73447,2754	Henry Baumgarten,

	Board Member
72401,1417	Frank Lawyer, Board Member
70260,306	A. Hays Busch, Administrator
72747,3126	Robert Clark, Treasurer
73007,173	William Smith, Assistant SysOp, NewsLetter Editor
72740,66	Howard Sweet, Secretary-Designate

The meeting started at about 10:15, the Chaircritter showed up at 10:25, and came to order (with the appearance of Henry) at about 10:35 EST.

Matters dealt with were:

The Secretary Problem

In the interim since the January meeting, Frank Lawyer had done a heroic job of arm-twisting, and persuaded Howard Sweet to take over the duties of the Secretary. Most of the first half of the meeting was involved in the Chaircritter trying to get a straight & explicit answer (on the record) from Howard that he was indeed willing to be USUS Secretary -- he was and is, and with defining what the Secretary needs to do -- make up a paper Book of Minutes, and how that might be accomplished. Having done all of the above, the Board congratulated Howard on becoming Secretary, and moved on to:

The President Problem

In the interim since the January meeting, Alex Kleider had accepted the job of USUS President, and continually asked for direction on what duties were expected of the person in that post. After very little discussion, it was determined that the most important tasks for the President to accomplish in 1989 were: 1) Increase USUS Membership; 2) Form an IBM/PC SIG; and 3) Arrange for a 1989 General Meeting.

This was agreed to by unanimous vote of the Board and Officers present, and the President was so instructed.

The JPAM Question

This was Old Business, continued from the January meeting, about what kind of a relationship should be formed with the publisher of the Journal of Pascal, Ada, and Modula-2. Henry reported that he had talked with both Prof. Weiner, the Editor, and with the Publisher, who had made an offer of terms which Henry posted in Section 8. He evinced a willingness to continue discussions with the Publisher, but not without direction from the Board on what directions to go, and what positions to take. Since several of the Board Members and Officers (notably including the Chaircritter) had not had an opportunity to read the offers, the matter was tabled pro tem, and all agreed to read the offers off-line and get back to Henry as soon as possible.

The meeting then broke up (in several senses -- the Chaircritter's computer printed garbage & refused to straighten out) with a vague agreement to meet again soon, and certainly on the 2nd Tuesday of March.

The Special Meeting of the Board on CompuServe was adjourned just before midnight EST on February 7th, 1989.

Minutes submitted by:
Samuel B. Bassett

WDS Environment (more background)
By William D. Smith

As I stated earlier, I didn't have room to include my string ops unit (StrOps_U) in this NewsLetter. I will be putting it in the next NewsLetter. In the NewsLetter after that I will be talking about my operating system interface unit (OpSys_U). This unit provides an interface to the system. With very few exceptions, none of my programs or other units use any of the system units. The operating system interface unit does require the interface of the units Kernal, DirInfo, FileInfo, SysInfo, Attribute, ScreenOps and Transfer (from Pecan) to compile. You may want to review the interface of these units before then.

For development work, I use a Sage IV with a 40 megabyte hard disk and 4 meg of memory. The 4 meg of memory provides a enough room for a large enough external code pool to put all the code in memory at once and also leaves enough for a 7700 block ramdisk.

The target environment consists of a version of the p-System (AOS, IV.13, IV.21 or IV.22) and a Wyse 50 or 60 terminal.

These environments have influenced the way I have designed the units and organized the files in what I call the "WDS Environment". To support the different versions of the p-System, I have tried to isolate both system and hardware dependant stuff in a few units.

The following unit is used so that I don't have problems like those in the system units (DirInfo, FileInfo, etc). In each of these units, there is a type for a date record (also LongString). The structure is the same but the names are different. This means, assigning a date of one type to the other is not allowed. You have to use a variant record or the pmachine instruction to do it.

WDS Globals Unit

By William D. Smith

This unit contains constants, types, a few variables and one procedure used by all my other programs and units. Two of the constants (`Vc_Glbs_U`, `Vs_Glbs_U`), one variable (`Vv_Glbs_U`) and the procedure (`Ck_Version`) are used for version control. Version control is only implemented for IV.22 since it depends on the order in which units are initialized,guaranteed by IV.22 and AOS, but not the other versions. AOS provides version control at the system level so it need not be done by the programmer. There is a bug in QuickStart which initializes units in the wrong order, so I do not use QuickStart (don't miss it since all the programs I use are resident in a sub volume on my ramdisk).

How does version control work? In the initialization section of each unit, the version variable (`Vv_Glbs_U` in this case) is set to the version constant (`Vc_Glbs_U`). The compiler embeds the value of the constant in the code (5 for this unit). When the code is executed at runtime, the variable is set to 5. The client unit or program makes a call to `Ck_Version` passing it the same constant and variable. As before when the compiler compiles the call, it embeds the constant 5 in the code. `Ck_Version` compares the variable and constant and if they are not the same, give a message and aborts the program. Now when the interface of the unit changes, the version is incremented. When the call to `Ck_Version` is compiled, the constant 6 is put in the code. When the program is run, if an old version of the unit is used (with constant 5 in its initialization section), the version variable and constant will not be the same and the program will abort.

The constants in this unit are mostly error codes and the value of `Null` (a blank or not used value).

The types are for the common lengths of strings I use (all odd length since the compiler allocates even number of bytes for a string and the length byte is not counted). I declare `CharSet` so that when variables of type "set of char" are declared I can use this instead. `CharSet` uses 8 words, "set of char" uses 16 words. `CharSet` only supports 7 bit characters. `OnOff` is used for a stack of on and off values. `YesNo` is a four way toggle. `FibPtr` is used to handle dynamic file allocation such as "array of file". More when I present my file I/O units. `DateRec` and `TimeRec` are declared here so that the different units which will use dates and times use the same record type. It matches the structure of the system date and time record.

The variables are used for very low level inter-unit communication (`ScUsed`, `Debug`) and variables (`Closed`, `NullTad`) which should be constants but are not allowed by Pascal.

```
{ WDS Globals unit                    [1.13] --- 09 Mar 88 } { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1988, 1989,  All rights reserved.                 }

{ File:         Glbs_U.Text                Version 1.13    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) 1988 to 1989.

  System:       Power System version IV.2.2

  Compiler:     Power System Pascal Compiler

  Keywords:     WDS Glbs_U Globals Unit

  Description:  WDS globals unit.  This unit contains the stuff used by the other
                WDS units.

  Change log:   (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
09 Mar 88   WDS  1.13  Added Direction and CmpType.
14 Feb 88   WDS  1.12  Added FibPtr, Closed and ScUsed.
08 Feb 88   WDS  1.11  Added M_NoHeap.
06 Feb 88   WDS  1.10  Added CopyRight message and M_ errors.
23 Oct 87   WDS  1.09  Added Str_1.
20 Oct 87   WDS  1.08  Added Vs_Glbs_U.
16 Sep 87   WDS  1.07  Added SysFib type.
20 Aug 87   WDS  1.06  Put in a readln after message.
16 Jul 87   WDS  1.05  Put in Ck_Version.
12 Jun 87   WDS  1.04  Added YesNo, YnSet and OoSet.
02 Jun 87   WDS  1.03  Added Byte.
28 May 87   WDS  1.02  Added Str_31.
27 May 87   WDS  1.01  Added TimeRec, TadRec, and OnOff.  NullDate to NullTad.
08 May 87   WDS  1.00  Started creating this unit.
}
{$I VERSION.TEXT}  { Declares conditional compilation flags }

unit  Glbs_U;

interface  {$ Glbs_U [1.13] 09 Mar 88 }

const Vc_Glbs_U   = 5;          { 14 Feb 88 }
      Vs_Glbs_U   = 'Glbs_U';

      CopyRight   =
       '(c) Copyright 1983 to 1989 by William D. Smith. All rights reserved';
      Null        = - 1;        { Blank or not used value }

      M_NoError   = 0;          { No error or message }
                                { Errors 1 to 31 are I/O errors and are
                                  the same as the system I/O errors. }

      M_Empty     = 43;         { File is empty }
      M_Bof       = 44;         { At begining of file }
      M_Eof       = 45;         { At end of file }
      M_Unknown   = 46;         { Unknown I/O error }
      M_NoHeap    = 47;         { No more room on the heap }
      M_ExecErr   = 48;         { Errors 49 to 79 are execution errors and
                                  are the system execution errors offset by
                                  48.  There are 32 possible. }

      M_UserErr   = 80;         { Errors 80 to 255 are user defined errors }
```

```
type  Byte        = 0..255;

      Str_1       = string [1];
      Str_3       = string [3];
      Str_5       = string [5];
      Str_7       = string [7];
      Str_9       = string [9];
      Str_15      = string [15];
      Str_23      = string [23];
      Str_31      = string [31];
      Str_63      = string [63];
      Str_81      = string [81];
      Str_133     = string [133];
      Str_255     = string [255];

      CharSet     = set of ' '..'~';  { 7 bit chars, 8 words in set }

      OnOff       = (On, Off, Pop, Toggle, Show);  { Stack values }
      OoSet       = set of OnOff;

      YesNo       = (Neither, Yes, No, Only);  { Three or four way toggle }
      YnSet       = set of YesNo;

      CmpType     = (Lt,       { Less then }
                     Eq,       { Equal }
                     Gt        { Greater then }
                    );

      FibPtr      = ^integer;       { Initialize to Closed }

      Direction   = (F_What,   { First }
                     P_What,   { Previous }
                     C_What,   { Current }
                     N_What,   { Next }
                     L_What    { Last }
                    );

      DateRec     = packed record
                      Month : 0..12;
                      Day   : 0..31;
                      Year  : 0..100;
                    end { DateRec };

      TimeRec     = packed record
                      Min   : 0..59;
                      Hour  : 0..24;
                    end { TimeRec };

      TadRec      = record
                      D : DateRec;
                      T : TimeRec;
                    end { TadRec };

var   Vv_Glbs_U   : integer;
      ScUsed      : boolean; { Set when low level stuff uses screen (Rv_U) }
      Debug       : boolean; { Controlled by T_Io_U }
      Closed      : FibPtr;
      NullTad     : TadRec;

  procedure Ck_Version (Vv, Vc : integer;  Vs_User, Vs_Used : Str_15);
  { This procedure compares Vv and Vc.  If they are different, an error message
    is output, the procedure waits for a return to be typed and then the program
    halts.  Vv for each unit should be the first variable.  Vs_User is the user
    program or unit name.  Vs_Used is the used unit name.

  }
```

```
implementation
  procedure Ck_Version { (Vv, Vc : integer;  Vs_User, Vs_Used : Str_15) };
  begin
  {$B IV22+}
    if Vv <> Vc then
      begin
        write ('ERORR: "', Vs_User, '" uses wrong version of "',
                           Vs_Used, '" <ret> ');
        readln;
        exit (program);
      end { if };
  {$E IV22+}
  end { Ck_Version };

begin { Glbs_U }
  Vv_Glbs_U := Vc_Glbs_U;

  ScUsed := false;
  Debug := false;

  Closed := nil;

  fillchar (NullTad, sizeof (NullTad), 0);
  NullTad .T .Hour := 24;      { To match system time standards }

  *** ;
end {$Q- Glbs_U }.
```

*The following three Modula2 modules where submitted by **Alex Kleider**. They show the use of a stand alone module, **ListOps**, both the definition and implementation part (similar to a p-System unit) and a test module, **TestListOps** (similar to a p-System program). As you are reading, note the use of the conditional compilation features of Modula2 ($VAR, $SET, $IF and $END statements) for embedding test code in the modules . Alex has another module which handles lists implemented as files stored on disk. The TestListOps presented here was derived from the module to test that and the messages make references to files. When it says file, think list. These modules where developed with Volition's Modula2 and later ported to Senic's Modula2 on a Stride. When "•••" ends a line and begins the next line, these two lines must be combined and the "•••"s removed to compile.*

MODULE ListOps (definition)

by Alex Kleider

```
(* ListOps : definition of doubly linked circular list module.
   (c) copyright by A. Kleider, 1984-89. All rights reserved.

   An amalgamation of the original <lists> module (developed in 84) and the subsequently (Aug 85)
   developed circular doubly linked list. Implemented as a doubly linked circular list that can serve as
   a stack, a queue, or any sort of a list.

   Before a variable of type tList can be used it must be initialized with <initList>. When finished
   with its use, do a <freeList>.

   The client is strongly advised to use each freshly initialized <tList> as one and only one of the
   following and restrict him/herself to the appropriate associated procedures:

                        { insertLeft/Right     { headPosition
   (R)ound List         { putNode          }   { listPosition      }
                        { appendLists      }   { delete            }
                                               { getNode           }
   (O)rdered List    }                    }   { traverseLeft/Right }  { init/freeList
                     }                                                 { empty
                     }  { place                                        { headPosition
```

```
                    } { mergeLists                          }
                    }                                       }
(P)riority Queue    }                       }               }
                                            }   removeQ     }
(Q)ueue                 insertQ             }               }
                                                            }
(S)tack                 push                                }
                        pop
```

When using (round) lists, the client must guard against traversing (<traverseLeft> or <traverseRight>) and empty list. Also watch that you don't unbeknowingly go traversing past the head either to the left from head to tail or to right from tail to head. A similar problem can arise with the inserts.

If using an ordered list or a priority queue: the client module will have to define a procedure of type vtCompare. An example follows:

```
PROCEDURE compare ( itemA, itemB : ARRAY OF WORD ) : INTEGER;
VAR pA, pB : POINTER TO rtItem (* the data structure to be listed *);
BEIN
  pA := ADR ( itemA );  pB := ADR ( itemB );
  IF pA^.<key>    < pB^.<key> THEN RETURN -1;
  ELSIF pA^.<key> = pB^.<key> THEN RETURN 0;
  ELSE RETURN 1;
  END;
END compare;
```

As in all generic modules, there is little in the way of type checking and the client must take responsibility for never using any data type in conjunction with a particular list except for the type with which that list was initialized. My personal view/bias is that this is a responsibility that should not be beyond the abilities of those who might become clients!

```
*)
(*$PRINTERRORS := TRUE;*)
DEFINITION MODULE ListOps;
FROM SYSTEM IMPORT WORD;
EXPORT QUALIFIED  tList, tListPosition, vtCompare, initList, freeList, empty, headPosition,
                  push, pop, insertQ, removeQ, delete, getNode, putNode, place,
                  traverseLeft, traverseRight, insertLeft, insertRight, mergeLists, appendLists;
TYPE  tList; tListPosition;
      vtCompare = PROCEDURE ( ARRAY OF WORD, ARRAY OF WORD ) : INTEGER;
PROCEDURE initList ( VAR list : tList;  aItem : ARRAY OF WORD );
(*  initializes the data structure as an empty linked list/stack/queue *)
PROCEDURE  freeList ( VAR list : tList;  aItem : ARRAY OF WORD );
(*  Releases the memory used by the <list> structure.  Uses the data item, <aItem>, parameter just as
    a holder. *)
PROCEDURE empty ( list : tList ) : BOOLEAN;
(*  returns empty if the linked list is empty *)
PROCEDURE headPosition ( list : tList ) : tListPosition;
(*  Returns the head of the list.  The list head is initially defined as the first item inserted into the list.
    It can subsequently be changed by deletion, pushing, poping, removeQ, etc. *)
PROCEDURE push ( aItem : ARRAY OF WORD;  VAR stack : tList );
(*  inserts aItem to the top/head/front of the stack/queue/list onto the top of the list, replacing the head
    *)
```

```
PROCEDURE pop ( VAR stack : tList;  VAR aItem : ARRAY OF WORD );
(*  pops off the top; same as removeQ *)

PROCEDURE getNode ( list : tList;  loc : tListPosition;  VAR aItem : ARRAY OF WORD );
(*  pulls the data out of the list item at loc and into aItem list is not changed *)

PROCEDURE putNode ( aItem : ARRAY OF WORD;  VAR list : tList;  loc : tListPosition );
(*  converse of getListData: replaces the data at loc with aItem length of <list> remains unchanged but
    the data at <loc> ofcourse is. *)

PROCEDURE insertQ ( aItem : ARRAY OF WORD;  VAR q : tList );
(*  inserts aItem at the bottom/end/rear of the stack/queue/list *)

PROCEDURE removeQ ( VAR q : tList;  VAR aItem : ARRAY OF WORD );
(*  same as a pop *)

PROCEDURE insertRight ( aItem : ARRAY OF WORD;  VAR list : tList;
                        VAR loc : tListPosition );
(*  Inserts aItem into the linked list to the right of the item at loc.  <loc> takes on the new position.*)

PROCEDURE insertLeft ( aItem : ARRAY OF WORD;  VAR list : tList;
                       VAR loc : tListPosition );
(*  If <list> is <empty>, then simply adds <aItem> to the list; Otherwise Inserts <aItem> into the
    linked list to the left of the item at loc, <loc> takes on the new position. <listHead> remains
    unchanged so <insertLeft> relative to head of list is same as <insertQ>, not <push>. *)

PROCEDURE delete ( VAR list : tList;  VAR loc : tListPosition;
                   VAR aItem : ARRAY OF WORD );
(*  Deletes the item at loc from the linked list. The data in the deleted item is left in aItem. If the list
    head is deleted then list head is reset to the item that was on its right.  <loc> is set to position of
    item to right of the one deleted. *)

PROCEDURE traverseRight ( list : tList;  VAR loc : tListPosition;
                          VAR aItem : ARRAY OF WORD;  VAR bLastItem : BOOLEAN );
(* Traverses the linked list, right to left; reads the data at loc into aItem and then resets loc to the next
    item in the list.  If the <aItem> read is the last in the list then <lastItem> will be set to TRUE, else it
    is false. *)

PROCEDURE traverseLeft ( list : tList;  VAR loc : tListPosition;
                         VAR aItem : ARRAY OF WORD;  VAR bFirstItem : BOOLEAN );
(*  traverses the linked list, left to right; reads the data at loc into aItem and then resets loc to the next
    item in the list.  If the <aItem> read is the first item in the list (i.e. the head) then <bFirstItem> will
    be set TRUE, else it is set FALSE. *)

PROCEDURE mergeLists ( listA, listB : tList;  VAR newList : tList;
                       compare : vtCompare;  aA, aB : ARRAY OF WORD );
(* Assumed: listA & listB are ordered lists; newList is already initialized.  <listA> and <listB> are left
    as is.  <newList> will be ordered combination of both listA & listB.  aA & aB serve only as
    holders, they must be of appropriate type. *)

PROCEDURE appendLists ( listA, listB : tList;  VAR newList : tList;
                        aItem : ARRAY OF WORD );
(* <listA> & <listB> remain unchanged. <newList> <== listA plus listB.  <newList> is assumed to
    have been initialized.  <aItem> is only a holder and must be of appropriate type. *)

PROCEDURE place ( aItem : ARRAY OF WORD;  VAR list : tList;  unique : BOOLEAN;
                  VAR aItemB : ARRAY OF WORD;  compare : vtCompare );
(*  places aItem into the ordered list; if the <aItem>'s key (as detected by <compare>) is already in the
    list, then further action is determined by the boolean parameter <unique>:  if TRUE: the data item
    (with that key) that was in the list is put into <aItemB> and replaced by <aItem>;  if FALSE: there
    will end up being two data items in the list, both sharing an identical key.  NOTE: aItemB and
```

aItem MUST BE TWO SEPARATE VARIABLES of identical type. DO NOT USE THE SAME VARIABLE FOR BOTH PARAMETERS. *)

(* The number of items allowed in a list is limited by memory availability and by MAXCARD (i.e. 64K). (This should not pose any serious problems; if it does, the implementation can be recompiled with one type declaration modified to have a LONGCARD CARDINAL field.) *)

END ListOps.

MODULE ListOps (implementation)

by Alex Kleider

(* IMPLEMENTATION of ListOps MODULE
An amalgamation of the original <lists> module (developed in 84) and the subsequently (Aug 85) developed circular doubly linked list.

To give credit where credit be due: let it be known that the subjects of stacks, queues and linked lists were studied from the 1981 edition of the book "Data Structures Using Pascal" by Aaron M. Tenenbaum & Moshe J. Augenstein published by Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632. There are however many data structure text books that cover this material; this just happens to be the one available at the time development of these routines was begun (in 1984.)

<AdrsPerWord> was originally imported from Volition System's <SystemTypes> Module which isn't provided with the ScenicSoft system and hence is just declared a Constant of 2.

The DL (for debug listOps) compile time option is/was for debugging this module.

The DC (for debug client) compile time option is included with the expectation that a version with this option set to TRUE would be used during program development but that a finished product would import a version compiled with the option set to FALSE. The error handling procedure (errorTerminates) is set up the way it is to allow others to reimplement it as one that will report the nature of the error to the user and then allow the user to set the function result to either TRUE (indicating a desire to terminate the process) or FALSE (indicating a desire to carry on). Of course the response/result could be ignored by the client program, or this procedure could override what the client user wants and cause the program or process to halt regardless, depending ofcourse on the nature of the error.

*)

```
$VAR DL : BOOLEAN;
$SET "Include listOps debugging aids? " DL
$VAR DC : BOOLEAN;
$SET "Include client debugging aids? " DC

IMPLEMENTATION MODULE ListOps;

FROM SYSTEM IMPORT
        ADDRESS, ADR, WORD,
        CARDTOADDR, TSIZE;

FROM Storage IMPORT
        ALLOCATE, DEALLOCATE;

$IF DC OR DL THEN
FROM InOut IMPORT
        WriteLn, WriteString, WriteHex, Read;
$END;

CONST
    AdrsPerWord = 2;

TYPE
    tListPosition = POINTER TO rtNode;

    rtNode = RECORD
      pInfo : ADDRESS;
      pLt, pRt : tListPosition;
    END; (*rtNode*)

    tList = POINTER TO rtList;

    rtList = RECORD
      pHead : tListPosition;
      nInfoSize : ADDRESS;
      nSize : CARDINAL;
    END; (*rtList*)
```

```
$IF DL THEN
TYPE
    rtDatum = RECORD
      name : ARRAY [ 0 .. 27 ] OF CHAR;
      number : ARRAY [ 0 .. 13 ] OF CHAR;
    END;

VAR
    list : tList;
    rList : rtList;
    rNode : rtNode;

    datum : rtDatum;

    positionA,
    positionB,
    positionC : tListPosition;

    waitChar : CHAR;

PROCEDURE show
        ( subject : ARRAY OF CHAR;
          value : ARRAY OF WORD );
VAR i : CARDINAL;
BEGIN
 WriteLn;  WriteString ( subject );
 FOR i := 0 TO HIGH ( value ) DO
  WriteHex ( CARDINAL ( value [ i ] ), 5 );
 END;
END show;
$END;

PROCEDURE move
        ( A, B : ADDRESS;
          nMemoryLocations : LONGCARD );
(* A "smart" procedure which does a <move-
   Left> or <moveRight> as appropriate. *)
VAR i : LONGCARD;
      n : ADDRESS;
      step : LONGINT;

BEGIN
 IF ( A = B ) THEN RETURN; END;
 IF B < A THEN
  step := AdrsPerWord;
 ELSE (* A < B *)
  step := - AdrsPerWord;
  n := ( nMemoryLocations - AdrsPerWord );
  A := A + n;
  B := B + n;
 END;
 i := 0;
 REPEAT
  B^ := A^;
  A := A + ADDRESS ( step );
  B := B + ADDRESS ( step );
  i := i + AdrsPerWord;
 UNTIL i >= nMemoryLocations;
END move;

PROCEDURE size
        ( aWords : ARRAY OF WORD ) :
        ADDRESS;

BEGIN
 RETURN CARDTOADDR
       (AdrsPerWord * (HIGH (aWords) + 1));
END size;
```

```
$IF DC THEN

PROCEDURE errorTerminates ( errorNumber : CARDINAL ) : BOOLEAN;
VAR string : ARRAY [ 0 .. 59 ] OF CHAR;  ch : CHAR;
BEGIN
 CASE errorNumber OF
    0 : string := ('Failure to create Semaphores in <termIO> initialization.      ');
 |  1 : ;
 |  2 : ;
 |  10 : string := ('Parameter size mismatch in <ListOps.freeList>.             ');
 |  11 : string := ('Attempt to <ListOps.getNode> from an empty list.           ');
 |  12 : string := ('Parameter size mismatch in <ListOps.getNode>.              ');
 |  13 : string := ("<ListOps.delete> received an unallocated <loc> pointer.    ");
 |  14 : string := ('Parameter size mismatch in <ListOps.putNode>.              ');
 |  15 : string := ('Parameter size mismatch in <ListOps.insertRight>.          ');
 |  16 : string := ('Parameter size mismatch in <ListOps.insertLeft>.           ');
 |  17 : string := ('Parameter size mismatch in <ListOps.delete>.               ');
 |  18 : string := ('Attempt to delete from empty list in <ListOps.delete>.     ');
```

```
  | 19 : string := ('Attempt to <ListOps.traverseRight> through an empty list.   ');
  | 20 : string := ('<ListOps.traverseRight> given <NIL> pointer.               ');
  | 21 : string := ('Parameter size mismatch in <ListOps.traverseRight>.        ');
  | 22 : string := ('Attempt to <ListOps.traverseLeft> through an empty list.   ');
  | 23 : string := ('<ListOps.traverseLeft> given <NIL> pointer.               ');
  | 24 : string := ('Parameter size mismatch in <ListOps.traverseLeft>.        ');
  | 25 : string := ('Parameter size mismatch in <ListOps.mergeLists>.          );
  | 26 : string := ('Parameter size mismatch in <ListOps.appendLists>.         ');
  | 27 : string := ('Parameter size mismatch in <ListOps.place>.               ');
  ELSE
   (* do nothing; undefined as yet *);
  END; (* case *)
  WriteLn;  WriteString ( string );  WriteLn;  WriteString ( "any key to continue" );
  Read ( ch );
  RETURN TRUE;
END errorTerminates;

PROCEDURE sizeOK ( aItem : ARRAY OF WORD;  list : tList ) : BOOLEAN;
BEGIN
  RETURN size ( aItem ) = list^.nInfoSize;
END sizeOK;

$END;
```

```
PROCEDURE newNode
        ( list : tList ) : tListPosition;

VAR pNode : tListPosition;
      adr : ADDRESS;

BEGIN
  ALLOCATE ( pNode, TSIZE ( rtNode ) );
  ALLOCATE ( pNode^.pInfo, list^.nInfoSize );
  adr := pNode^.pInfo;
  pNode^.pLt := NIL;
  pNode^.pRt := NIL;
$IF DL THEN
  rNode := pNode^;
  show ("<newNode> returns:", pNode );
  show (" and rNode is:", rNode );
$END;
  RETURN pNode;
END newNode;

PROCEDURE initList
        ( VAR list : tList;
          aItem : ARRAY OF WORD );

(* initializes the data structure as an empty linked
   list/stack/queue *)

BEGIN
  ALLOCATE ( list, TSIZE ( rtList ) );
  list^.pHead := NIL;
  list^.nInfoSize := size ( aItem );
  list^.nSize := 0;
END initList;

PROCEDURE  freeList
        ( VAR list : tList;
          aItem : ARRAY OF WORD );

(* Releases the memory used by the <list>
   structure.  Uses the data item, <aItem>,
   parameter just as a holder. *)

BEGIN
$IF DC THEN
  IF NOT sizeOK ( aItem, list ) THEN
    IF errorTerminates ( 10 ) THEN ;
    END;
  END;
$END
  WHILE NOT empty ( list ) DO
    pop ( list, aItem );
  END; (* while nSize # 0 *)
  DEALLOCATE ( list, TSIZE ( rtList ) );
END freeList;

PROCEDURE empty ( list : tList ) : BOOLEAN;
(* returns empty if the linked list is empty *)
BEGIN
  RETURN list^.nSize = 0;
END empty;
```

```
PROCEDURE headPosition
        ( list : tList ) : tListPosition;
(* Returns the head of the list.  The list head is
   initially defined as the first item inserted into
   the list.  It might be changed by deletion,
   pushing, poping, removeQ, etc. *)
BEGIN
 RETURN list^.pHead;
END headPosition;

PROCEDURE push
        ( aItem : ARRAY OF WORD;
          VAR stack : tList );
(* inserts aItem to the top/head/front of the
   stack/queue/list onto the top of the list,
   replacing the head  *)
VAR pLoc : tListPosition;
BEGIN
 insertLeft ( aItem, stack, stack^.pHead );
END push;

PROCEDURE pop
        ( VAR stack : tList;
          VAR aItem : ARRAY OF WORD );
(* removes the data from the top/head/front of the
   stack/queue/list and puts into aItem, pops off
   the top; same as removeQ *)
VAR pLoc : tListPosition;
BEGIN
 delete ( stack, stack^.pHead, aItem );
END pop;

PROCEDURE getNode
        ( list : tList;
          loc : tListPosition;
          VAR aItem : ARRAY OF WORD );
(* pulls the data out of the list item at loc and into
   aItem  *)
BEGIN
$IF DC THEN
 IF list^.nSize = 0 THEN
  IF errorTerminates ( 11 ) THEN ;
  END;
 END;
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 12 ) THEN ;
  END;
 END;
$END;
$IF DL THEN
 show ("/a move in getNode. loc", loc );
$END
 move ( loc^.pInfo,
 ADR ( aItem ), list^.nInfoSize );
END getNode;

PROCEDURE putNode
        ( aItem : ARRAY OF WORD;
          VAR list : tList;
          loc : tListPosition );
(* converse of getNode: replaces the data at loc
   with aItem  *)
BEGIN
$IF DC THEN
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 14 ) THEN ;
  END;
 END;
$END
 move ( ADR ( aItem ),
 loc^.pInfo, list^.nInfoSize );
END putNode;

PROCEDURE insertQ
        ( aItem : ARRAY OF WORD;
          VAR q : tList );
(* inserts aItem at the bottom/end/rear of the
   stack/queue/list *)
VAR pLoc : tListPosition;
BEGIN
 pLoc := q^.pHead;
 insertLeft ( aItem, q, pLoc );
END insertQ;

PROCEDURE removeQ
        ( VAR q : tList;
          VAR aItem : ARRAY OF WORD );
(* same as a pop *)
VAR pLoc : tListPosition;
BEGIN
 pLoc := q^.pHead;
 delete ( q, pLoc, aItem );
END removeQ;

PROCEDURE insertRight
        ( aItem : ARRAY OF WORD;
          VAR list : tList;
          VAR loc : tListPosition );
```

```
(* Inserts aItem into the linked list to the right of
   the item at loc.  <loc> takes on the new
   position.  *)
VAR pNew : tListPosition;
BEGIN
$IF DC THEN
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 15 ) THEN ;
  END;
 END;
$END
 pNew := newNode ( list );
 IF list^.nSize = 0 THEN
  pNew^.pLt := pNew;
  pNew^.pRt := pNew;
  list^.pHead := pNew;
 ELSIF list^.nSize = 1 THEN
  pNew^.pLt := loc;
  pNew^.pRt := loc;
  loc^.pLt := pNew;
  loc^.pRt := pNew;
 ELSE
  pNew^.pLt := loc;
  pNew^.pRt := loc^.pRt;
  loc^.pRt^.pLt := pNew;
  loc^.pRt := pNew;
 END;
 loc := pNew;
 INC ( list^.nSize );
 putNode ( aItem, list, loc );
END insertRight;

PROCEDURE insertLeft
        ( aItem : ARRAY OF WORD;
          VAR list : tList;
          VAR loc : tListPosition );
(* Inserts aItem into the linked list to the left of
   the item at loc, <loc> takes on the new
   position. <listHead> remains unchanged so
   <insertLeft> relative to head of list is same as
   <insertQ>, not <push>. *)
VAR pNew : tListPosition;
BEGIN
$IF DC THEN
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 16 ) THEN ;
  END;
 END;
$END
 pNew := newNode ( list );
 IF list^.nSize = 0 THEN
  pNew^.pLt := pNew;
  pNew^.pRt := pNew;
  list^.pHead := pNew;
 ELSIF list^.nSize = 1 THEN
  pNew^.pLt := loc;
  pNew^.pRt := loc;
  loc^.pLt := pNew;
  loc^.pRt := pNew;
 ELSE
  pNew^.pLt := loc^.pLt;
  pNew^.pRt := loc;
  loc^.pLt^.pRt := pNew;
  loc^.pLt := pNew;
 END;
 loc := pNew;
 INC ( list^.nSize );
 putNode ( aItem, list, loc );
$IF DL THEN
 rNode := loc^;
 show ("In insertLeft, loc^ after putNode:",
       rNode );
$END;
END insertLeft;

PROCEDURE delete
        ( VAR list : tList;
          VAR loc : tListPosition;
          VAR aItem : ARRAY OF WORD );
(* Deletes the item at <loc> and leaves it in
   <aItem>. *)
VAR rNode : rtNode;  bHead : BOOLEAN;
BEGIN
$IF DL THEN
 show ("Entering delete with loc:", loc );
$END;
$IF DC THEN
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 17 ) THEN ;
  END;
 END;
(* There are 3 special situations:
   The FIRST is an error condition if the list is
   empty. *)
 IF ( list^.nSize = 0 ) THEN
  IF errorTerminates ( 18 ) THEN ;
  END;
 END;
```

```
 IF ( loc = NIL ) THEN
  IF errorTerminates ( 13 ) THEN ;
  END;
 END;
$END
 IF ( list^.nSize = 0 ) OR ( loc = NIL ) THEN
  RETURN;
 END;
 rNode := loc^;
 IF loc = list^.pHead THEN
  bHead := TRUE;
 ELSE
  bHead := FALSE;
 END;
 getNode ( list, loc, aItem );
 DEALLOCATE ( loc, TSIZE ( rtNode ) );
(* The SECOND is situation in which there is
   only one item in the list.. *)
 IF list^.nSize = 1 THEN
  list^.pHead := NIL;
  loc := NIL;
 ELSE
(* Finally, all other situations can be handled the
   same way.. *)
  IF bHead THEN
   list^.pHead := rNode.pRt;
  END;
  loc := rNode.pRt;
  rNode.pLt^.pRt := rNode.pRt;
  rNode.pRt^.pLt := rNode.pLt;
 END;
 DEALLOCATE ( rNode.pInfo,
                  list^.nInfoSize );
 DEC ( list^.nSize );
END delete;

PROCEDURE traverseRight
        ( list : tList;
          VAR loc : tListPosition;
          VAR aItem : ARRAY OF WORD;
          VAR bLastItem : BOOLEAN );
(* Traverses the linked list, right to left;  reads the
   data at loc into aItem and then resets loc to the
   next item in the list.  If the <aItem> read is the
   last in the list then <lastItem> will be set to
   TRUE, else it is false. *)
BEGIN
$IF DC THEN
 IF list^.nSize = 0 THEN
  IF errorTerminates ( 19 ) THEN ;
  END;
 END;
 IF loc = NIL THEN
  IF errorTerminates ( 20 ) THEN ;
  END;
 END;
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 21 ) THEN ;
  END;
 END;
$END
 getNode ( list, loc, aItem );
 loc := loc^.pRt;
 IF loc = list^.pHead THEN
  bLastItem := TRUE;
 ELSE
  bLastItem := FALSE;
 END;
END traverseRight;

PROCEDURE traverseLeft
        ( list : tList;
          VAR loc : tListPosition;
          VAR aItem : ARRAY OF WORD;
          VAR bFirstItem : BOOLEAN );
(* traverses the linked list, left to right; reads the
   data at loc into aItem and then resets loc to the
   next item in the list.  If the <aItem> read is the
   first item in the list (i.e. the head) then
   <bFirstItem> will be set TRUE, else it is set
   FALSE;     *)
BEGIN
$IF DC THEN
 IF list^.nSize = 0 THEN
  IF errorTerminates ( 22 ) THEN ;
  END;
 END;
 IF loc = NIL THEN
  IF errorTerminates ( 23 ) THEN ;
  END;
 END;
 IF NOT sizeOK ( aItem, list ) THEN
  IF errorTerminates ( 24 ) THEN ;
  END;
 END;
$END
 getNode ( list, loc, aItem );
```

```
  IF loc = list^.pHead THEN
   bFirstItem := TRUE;
  ELSE
   bFirstItem := FALSE;
  END;
  loc := loc^.pLt;
END traverseLeft;

PROCEDURE mergeLists
        ( listA, listB : tList;
          VAR newList : tList;
          compare : vtCompare;
          aA, aB : ARRAY OF WORD );
(* Assumed: listA & listB are ordered lists;
   newList is already initialized. <listA> and
   <listB> are left as is. <newList> will be
   ordered combination of both listA & listB. aA
   & aB serve only as holders, they must be of
   appropriate type. *)
VAR xCompare : INTEGER;
      iA, iB : CARDINAL;
      pA, pB : tListPosition;
      bEndA, bEndB : BOOLEAN;
BEGIN
$IF DC THEN
 IF ( NOT sizeOK ( aA, listA ) ) OR
    ( NOT sizeOK ( aA, listB ) ) OR
    ( NOT sizeOK ( aA, newList ) ) OR
    ( NOT ( HIGH ( aA ) = HIGH ( aB ) ) )
 THEN
  IF errorTerminates ( 25 ) THEN ;
  END;
 END;
$END
 freeList ( newList, aB );
 initList ( newList, aB );
 IF ( listA^.nSize = 0 ) OR
    ( listB^.nSize = 0 ) THEN
  appendLists ( listA, listB, newList, aA );
  RETURN;
 END;
 pA := listA^.pHead;
 pB := listB^.pHead;
 traverseRight ( listA, pA, aA, bEndA );
 traverseRight ( listB, pB, aB, bEndB );
 LOOP
  xCompare := compare ( aA, aB );
  IF xCompare < 1 THEN
   insertQ ( aA, newList );
   IF bEndA THEN
    WHILE NOT bEndB DO
     insertQ ( aB, newList );
     traverseRight ( listB, pB, aB, bEndB );
    END;
    insertQ ( aB, newList );
    EXIT;
   ELSE
    traverseRight ( listA, pA, aA, bEndA );
   END;
  ELSE (* xCompare > 0 *)
   insertQ ( aB, newList );
   IF bEndB THEN
    WHILE NOT bEndA DO
     insertQ ( aA, newList );
     traverseRight ( listA, pA, aA, bEndA );
    END;
    insertQ ( aA, newList );
    EXIT;
   ELSE
    traverseRight ( listB, pB, aB, bEndB );
   END;
  END; (* ... ELSE xCompare > 0 *)
 END; (* LOOP *)
END mergeLists;

PROCEDURE appendLists
        ( listA, listB : tList;
          VAR newList : tList;
          aItem : ARRAY OF WORD );
(* <listA> & <listB> remain unchanged.
   <newList> <== listA plus listB. <newList> is
   assumed to have been initialized. <aItem> is
   only a holder and must be of appropriate type.
   *)
VAR i : CARDINAL;
      pPos : tListPosition;
      bEnd : BOOLEAN;
BEGIN
$IF DC THEN
 IF ( NOT sizeOK ( aItem, listA ) ) OR
    ( NOT sizeOK ( aItem, listB ) ) OR
    ( NOT sizeOK ( aItem, newList ) ) THEN
  IF errorTerminates ( 26 ) THEN ;
  END;
 END;
$END
```

```
 freeList ( newList, aItem );
 initList ( newList, aItem );
 pPos := listA^.pHead;
 FOR i := 1 TO listA^.nSize DO
  traverseRight ( listA, pPos, aItem, bEnd );
  insertQ ( aItem, newList );
 END;
 pPos := listB^.pHead;
 FOR i := 1 TO listB^.nSize DO
  traverseRight ( listB, pPos, aItem, bEnd );
  insertQ ( aItem, newList );
 END;
END appendLists;

PROCEDURE place
        ( aItem : ARRAY OF WORD;
          VAR list : tList;
          unique : BOOLEAN;
          VAR aItemB : ARRAY OF WORD;
          compare : vtCompare );
(* places aItem into the ordered list; if the data's
   key (as detected by <compare>) is already in
   the list, then the boolean parameter <unique>
   determins if that data item is put into aItemB
   and then replaced by aItem (when unique is
   true) or if the new data is simply added to the
   list (when unique is false). NOTE: aItemB
   and aItem MUST BE TWO SEPARATE
   VARIABLES OF IDENTICAL TYPE. DO
   NOT USE THE SAME VARIABLE FOR
   BOTH PARAMETERS. *)
 VAR xCompare : INTEGER;
       pLoc, pPrevious : tListPosition;
       bEnd : BOOLEAN;
 BEGIN
 $IF DC THEN
  IF ( NOT sizeOK ( aItem, list ) ) OR
     ( NOT sizeOK ( aItemB, list ) ) THEN
   IF errorTerminates ( 27 ) THEN ;
   END;
  END;
 $END
  IF list^.nSize = 0 THEN
   push ( aItem, list );
   RETURN;
  END;
  pLoc := list^.pHead;
  LOOP
   pPrevious := pLoc;
  traverseRight ( list, pLoc, aItemB, bEnd );
  xCompare := compare ( aItem, aItemB );
  IF xCompare = -1 THEN
   insertLeft ( aItem, list, pPrevious );
   IF pPrevious^.pRt = list^.pHead THEN
    list^.pHead := pPrevious;
   END;
   RETURN;
  ELSIF xCompare = 0 THEN
   IF unique THEN
    putNode ( aItem, list, pPrevious );
    RETURN;
   ELSE
    insertRight ( aItem, list, pPrevious );
    RETURN;
   END;
  ELSIF bEnd (* & xCompare = 1 *) THEN
   insertRight ( aItem, list, pPrevious );
   RETURN;
  END;
(* if xCompare = 1 but bEnd = FALSE then
   continue looping *)
 END; (* loop *)
END place;

BEGIN (* ListOps *)
$IF DL THEN
 initList ( list, datum );
 rList := list^;
 show ("After <initList>, list is", list );
 show ("and rList is", rList );
 datum.name := "                    ";
 datum.number := "          ";
 push ( datum, list );
 rList := list^;  rNode := rList.pHead^;
 show ("After 1ST <push>, list is", list );
 show ("rList is", rList );
 show ("and rNode is", rNode );
 Read ( waitChar );
 push ( datum, list );
 rList := list^;  rNode := rList.pHead^;
 show ("After 2ND <push>, list is", list );
 show ("rList is", rList );
 show ("and rNode is", rNode );
 show ("About to enter <freeList> with list",
        list );
 rList := list^;
 show ("and rList ", rList );
```

```
  freeList ( list, datum );
  Read ( waitChar );
```

```
$END;
END ListOps.
```

MODULE TestListOps
by Alex Kleider

```
MODULE TestListOps;

FROM SYSTEM IMPORT
        WORD, ADR, ADDRESS,
        ADDRTOLONG;

FROM Screen IMPORT
        ClearScreen, GotoXY;

FROM InOut IMPORT
        WriteString, WriteCard, Read,
        ReadString, ReadCard, WriteLongHex;

FROM Strings IMPORT
        CompareStr;

FROM ListOps IMPORT
        tList, tListPosition, initList, freeList,
        empty, headPosition, push, pop,
        getNode, putNode, insertQ, removeQ,
        insertLeft, insertRight, delete,
        traverseLeft, traverseRight, mergeLists,
        appendLists, place, vtCompare;

TYPE
    tListId = RECORD
      type : CHAR;
      pointer : tList;
      position : tListPosition;
    END;

    rtDatum = RECORD
      name : ARRAY [ 0 .. 27 ] OF CHAR;
      number : ARRAY [ 0 .. 13 ] OF CHAR;
    END;

    atId = ARRAY [ 0 .. 13 ] OF CHAR;

VAR
    aList : ARRAY [ 0 .. 9 ] OF tListId;
    i : CARDINAL;
    item : rtDatum;

PROCEDURE fileType
        ( type : CHAR; VAR id : atId );
BEGIN
 CASE type OF
   "R": id := "(round) List  ";
 | "O": id := "Ordered List  ";
 | "P": id := "Priority Queue";
 | "Q": id := "Queue       ";
 | "S": id := "Stack       ";
 ELSE
   id := "type not impl.";
 END; (* case *)
END fileType;

(* THE FOLLOWING ARE "DIRTY
   PROGRAMMING" TRICKS: *)

PROCEDURE nilPosition
        ( position : tListPosition ) : BOOLEAN;
BEGIN
 RETURN ( ADDRESS ( position ) = NIL );
END nilPosition;

PROCEDURE nilList
        ( list : tList ) : BOOLEAN;
BEGIN
 RETURN ( ADDRESS ( list ) = NIL );
END nilList;

PROCEDURE setPositionToNil
        ( VAR position : tListPosition );
VAR adr : POINTER TO ADDRESS;
BEGIN
 adr := ADR ( position );
 adr^ := NIL;
END setPositionToNil;

PROCEDURE setListToNil
        ( VAR list : tList );
VAR adr : POINTER TO ADDRESS;
BEGIN
 adr := ADR ( list );
 adr^ := NIL;
END setListToNil;

PROCEDURE writePointer
        ( pointer : tListPosition;
          n : CARDINAL );

TYPE
    rtTrick = RECORD
      CASE BOOLEAN OF
        TRUE : pointer : tListPosition;
      | FALSE : long : LONGCARD;
      END;
    END;

VAR rTrick : rtTrick;
```

```
BEGIN
 rTrick.pointer := pointer;
 WriteLongHex ( rTrick.long, n );
END writePointer;

PROCEDURE getRec
        ( VAR rec : rtDatum;
          x, y : CARDINAL );

BEGIN (* may require up to 71 characters *)
 GotoXY ( x, y );
   WriteString  ("....................................");
   WriteString  ("....................................\");
 GotoXY ( x, y );
 WriteString ("Enter Name: ");
 ReadString ( rec.name );
 WriteString ("; Enter number: ");
 ReadString ( rec.number );
END getRec;

PROCEDURE clearLine ( y : CARDINAL );
BEGIN
 GotoXY ( 0, y );
 WriteString ("                                    ");
 WriteString ("                                    ");
END clearLine;

PROCEDURE showRec
        ( rec : rtDatum;
          x, y : CARDINAL );

BEGIN (* may require up to 59 characters *)
 GotoXY ( x, y );
   WriteString  ("............................");
   WriteString  ("............................\");
 GotoXY ( x, y );
 WriteString ("Name: ");
 WriteString ( rec.name );
 WriteString ("; Number: ");
 WriteString ( rec.number );
END showRec;

PROCEDURE compare
        ( a, b : ARRAY OF WORD ) :
        INTEGER;

VAR pA, pB : POINTER TO rtDatum;
BEGIN
 pA := ADR ( a );
 pB := ADR ( b );
 RETURN ( CompareStr ( pA^.name,
                       pB^.name ) );
END compare;

PROCEDURE pickAList
        ( prompt : ARRAY OF CHAR;
          x, y : CARDINAL ) : CARDINAL;

(* Returns index of first active member of the
   array.  Out of range if none exists. *)

VAR i : CARDINAL;
    choice : CHAR;
    empty : BOOLEAN;

BEGIN
 empty := TRUE;
 FOR i := 0 TO 9 DO
  IF NOT nilList ( aList [ i ].pointer ) THEN
   empty := FALSE;
  END;
 END;

 IF empty THEN
  RETURN ( 9 + 1 );
 END;

 LOOP
  GotoXY ( x, y );
  WriteString ( prompt );
  Read ( choice );

  IF ( choice >= "0" ) AND
     ( choice <= "9" )  THEN
   i := ( ORD ( choice ) - ORD  ("0") );
   IF (NOT (nilList (aList [i].pointer ))) THEN
    RETURN i;
   END;
  END;
 END;(*loop*)
END pickAList;

PROCEDURE pickASpot ( ) : CARDINAL;

(* Finds an unused spot in the array. Returns out
   of range if none exists. *)

VAR i : CARDINAL;
BEGIN
 FOR i := 0 TO 9 DO
  IF nilList ( aList [ i ].pointer ) THEN
   RETURN i;
  END;(*if*)
 END;(*for*)

 RETURN ( 9 + 1 ); (* array is full *)
END pickASpot;

PROCEDURE listLists;
```

```
VAR i, y : CARDINAL;
      id : atId;
      empty : BOOLEAN;
BEGIN
 empty := TRUE;
 GotoXY ( 60,  5 );
 WriteString ("Active Lists:");
 GotoXY ( 60,  6 );
 WriteString ("============");
 y := 7;
 FOR i := 0 TO 9 DO
  IF aList [ i ].type # 0C THEN
   empty := FALSE;
   GotoXY ( 58, y );
   WriteCard ( i, 3 );
   fileType ( aList [ i ].type, id );
   WriteString (": ");
   WriteString ( id );
   IF NOT nilPosition (aList [i].position) THEN
    WriteString (" *");
   END;
   INC ( y );
  END; (*if*)
 END; (* for *)
 IF empty THEN
  GotoXY ( 62, y );
  WriteString ("none");
 ELSE
  INC ( y );
  GotoXY ( 60, y );
  WriteString ("* = marked");
 END;
END listLists;

PROCEDURE terminate
        ( x, y : CARDINAL;
          message : ARRAY OF CHAR );
VAR ch : CHAR;
BEGIN
 GotoXY ( x, y );
 WriteString ( message );
 WriteString (" Any key to continue. ");
 Read ( ch );
END terminate;

PROCEDURE newList;
VAR i : CARDINAL;
      ch : CHAR;
      datum : rtDatum;
BEGIN
 ClearScreen;
 listLists;
 i := pickASpot ();
 GotoXY ( 10, 10 );
 WriteString ("Beginning a new List:");
 GotoXY ( 10, 11 );
 WriteString ("====================");
 IF i > 9 THEN
  terminate ( 2, 13,
      "No room for list; must <giveUpList>.");
  RETURN;
 END;
 REPEAT
  GotoXY (  2, 13 );
  WriteString ("What type: R(oundList, ");
  WriteString ("O(rderedList, Q(ue, ");
  WriteString ("P(riorityQue, S(tack: ");
  Read ( ch );
  IF ORD ( ch ) >= ORD ( "a" ) THEN
   IF ORD ( ch ) <= ORD ( "z" ) THEN
    ch := CHAR ( ORD ( ch ) -
                ( ORD ("a") - ORD ("A") ) );
   END;
  END;
 UNTIL ( ORD ( ch ) >= ORD ("O") ) AND
        ( ORD ( ch ) <= ORD ("S") );
 aList [ i ].type := ch;
 initList ( aList [ i ].pointer, datum );
 setPositionToNil ( aList [ i ].position );
END newList;

PROCEDURE giveUpList;
VAR i : CARDINAL;
      ch : CHAR;
      rDatum : rtDatum;
BEGIN
 ClearScreen;
 listLists;
 GotoXY ( 10, 10 );
 WriteString ("Deleting an existing list:");
 GotoXY ( 10, 11);
 WriteString
    ("=========================");
 i := pickAList ("Which list to delete? ", 10, 13 );
 IF i > 9 THEN
  terminate ( 10, 15,
          "No list is currently active.");
 ELSE
```

```
    freeList ( aList [ i ].pointer, rDatum );
    aList [ i ].type := 0C;
    setListToNil ( aList [ i ].pointer );
    setPositionToNil ( aList [ i ].position );
    terminate ( 10, 15,
          "Specified list has been eliminated.");
  END;
END giveUpList;

PROCEDURE mergeAppend;

VAR a, b, c : CARDINAL;
      item, itemA, itemB : rtDatum;

BEGIN
  ClearScreen;
  listLists;
  GotoXY ( 10, 10 );
  WriteString ("Combining Lists");
  GotoXY ( 10, 11 );
  WriteString ("===============");
  GotoXY ( 5, 14 );
  WriteString ("Note that destination list must ");
  WriteString ("exist but will be reinitialized.");

  a := pickAList ("First List: ",          11, 16 );
  b := pickAList ("Second List: ",         10, 17 );
  c := pickAList ("Destination: ",         10, 18 );

  IF ( a > 9 ) OR ( b > 9 ) OR ( c > 9 ) THEN
    terminate ( 5, 21,
          "Not enough files currently active.");
    RETURN;
  END;

  IF ( aList [ a ].type = "R" ) &
     ( aList [ b ].type = "R" ) THEN
    appendLists ( aList [ a ].pointer,
                  aList [ b ].pointer,
                  aList [ c ].pointer, item );
    setPositionToNil ( aList [ c ].position );
    terminate ( 5, 21, "Lists were <appended>.");
  ELSIF ( ( aList [ a ].type = "O" ) OR
            ( aList [ a ].type = "P" ) ) &
          ( ( aList [ b ].type = "O" ) OR
            ( aList [ b ].type = "P" ) ) THEN
    mergeLists ( aList [ a ].pointer,
                 aList [ b ].pointer,
                 aList [ c ].pointer,
                compare, itemA, itemB );
    terminate ( 5, 21, "Lists were <merged>.");
  ELSE
    terminate ( 0, 21, "Lists were incompatable •••
          •••for merging or appending.");
  END;(*ifthenelsifelse*)
END mergeAppend;

PROCEDURE showSpecs;

VAR id : atId; i : CARDINAL;
BEGIN
  ClearScreen;
  listLists;
  i := pickAList (
     "Show specifications of which file? ", 5, 5 );

  IF ( i > 9 ) THEN RETURN; END;

  fileType ( aList [ i ].type, id );
  GotoXY ( 10, 10 );
  WriteString ("File #");
  WriteCard ( i, 1 );
  WriteString (" is of type ");
  WriteString ( id );
  GotoXY ( 10, 11 );
  WriteString ("-it is ");

  IF NOT empty ( aList [ i ].pointer ) THEN
    WriteString ("not ");
  END;

  WriteString ("empty.");
  GotoXY ( 10, 12 );
  WriteString ("-its <headPosition> is ");
  writePointer ( headPosition
                 ( aList [ i ].pointer ), 6 );
  GotoXY ( 10, 13 );
  WriteString ("-there is ");

  IF nilPosition ( aList [ i ].position ) THEN
    WriteString ("NO ");
  ELSE
    WriteString ("a ");
  END;

  WriteString ("marked position.");
  terminate ( 10, 15, "That's all!");
END showSpecs;

PROCEDURE addItem;

VAR i : CARDINAL;
      ch : CHAR;
      rec, recB : rtDatum;

BEGIN
  ClearScreen;
  listLists;
  GotoXY ( 10, 10 );
  WriteString ("Adding to an existing list:");
  GotoXY ( 10, 11);
  WriteString
```

```
      ("============================");
  i := pickAList ("Add to which list? ", 10, 13 );
  IF i > 9 THEN
    terminate ( 10, 15,
                "No list is currently active.");
    RETURN;
  ELSE
    getRec ( rec, 0, 15 );
    CASE aList [ i ].type OF
      "R": IF empty ( aList [ i ].pointer ) THEN
             push ( rec, aList [ i ].pointer );
           ELSE
             terminate ( 10, 17,
                   "Wrong file type.");
             RETURN;
           END;
    | "O",
      "P": place ( rec, aList [ i ].pointer,
                   FALSE, recB, compare );
    | "Q": insertQ ( rec, aList [ i ].pointer );
    | "S": push ( rec, aList [ i ].pointer );
    | 0C : terminate ( 20, 5, "This list is inactive.");
           RETURN;
    ELSE
      terminate ( 20, 5,
              "Something is very wrong!");
    END;(*case*)
    showRec ( rec, 1, 15 );
    terminate ( 10, 16,
          "The above item has been added.");
  END;
END addItem;

PROCEDURE removeItem;
VAR i : CARDINAL; ch : CHAR; rec : rtDatum;
BEGIN
  ClearScreen;
  listLists;
  GotoXY ( 10, 10 );
  WriteString ("Removing from an existing list:");
  GotoXY ( 10, 11);
  WriteString ("================");
  WriteString ("================");
  i := pickAList (
          "Remove from which list? ", 10, 13 );
  IF i > 9 THEN
    terminate ( 10, 15,
          "No list is currently active.");
    RETURN;
  ELSE
    CASE aList [ i ].type OF
      "R",
      "O": terminate ( 10, 15, "Wrong file type.");
           RETURN;
    | "P",
      "Q": removeQ ( aList [ i ].pointer, rec );
    | "S": pop ( aList [ i ].pointer, rec );
    ELSE
      terminate ( 10, 16, "File type error.");
    END;(*case*)
    showRec ( rec, 1, 15 );
    terminate ( 10, 16,
          "The above item has been removed.");
  END;
END removeItem;

PROCEDURE manipulate;
TYPE tLeftOrRight = ( left, right );
VAR locA, locB : tListPosition;
      a, b : rtDatum;
      direction : tLeftOrRight;
      i, nPastFirst : CARDINAL;
      ch : CHAR;
      lastItem, firstItem : BOOLEAN;
    PROCEDURE leftOrRight ( ) : tLeftOrRight;
    BEGIN
      LOOP
        Read ( ch );
        IF ( ch = 'L' ) OR ( ch = "l" ) THEN
          RETURN left;
        END;
        IF ( ch = 'R' ) OR ( ch = "r" ) THEN
          RETURN right;
        END;
      END;(*loop*)
    END leftOrRight;

BEGIN (* body of <manipulate> *)
  LOOP
    ClearScreen;
    listLists;
    GotoXY ( 10, 2 );
    WriteString ("Manipulation of list.");
    GotoXY ( 10, 3 );
    WriteString
          ("====================");
    i := pickAList (
        "Pick a list to manipulate: ", 8, 5);
```

```
IF i > 9 THEN
 terminate ( 8, 6,
      "Appropriate file doesn't exist.");
 EXIT;
END;

IF empty ( aList [ i ].pointer ) THEN
 terminate ( 2, 6, "File is empty, must •••
           •••add a record before traversing.");
 EXIT;
END;

GotoXY ( 8, 6 );
WriteString ("Traverse right or left? ");
direction := leftOrRight();
locB := headPosition ( aList [ i ].pointer );
nPastFirst := 0;

IF direction = left THEN
 LOOP
  locA := locB;
  traverseLeft ( aList [ i ].pointer,
                 locB, a, lastItem );
  GotoXY ( 8, 7 );
  WriteString ("Node follows:");
  showRec ( a, 2, 8 );
  GotoXY ( 10, 9 );
  WriteString ("                    ");
  GotoXY ( 10, 9 );
  WriteString ("Mark this location? ");
  Read ( ch );

  IF ( ch  = 'y' ) OR ( ch = "Y" ) THEN
   aList [ i ].position := locA;
  END;(*if*)

  IF firstItem THEN INC ( nPastFirst ); END;

  IF nPastFirst >1 THEN EXIT; END;
 END;(*loop*)
ELSE (* direction = right *)
 LOOP
  locA := locB;
  traverseRight ( aList [ i ].pointer,
                  locB, a, lastItem );
  GotoXY ( 8, 7 );
  WriteString ("Node follows:");
  showRec ( a, 2, 8 );
  GotoXY ( 10, 9 );
  WriteString ("                    ");
  GotoXY ( 10, 9 );
  WriteString ("Mark this location? ");
  Read ( ch );

  IF ( ch  = 'y' ) OR ( ch = "Y" ) THEN
   aList [ i ].position := locA;
  END;(*if*)

  IF lastItem THEN EXIT; END;
 END;(*inner loop*)
END;

IF NOT nilPosition (aList [i].position) THEN
 GotoXY ( 10, 12 );
 WriteString ("There is a marked node.");
 GotoXY ( 9, 14 );
 WriteString ("0: Continue w/o any action.");
 GotoXY ( 9, 15 );
 WriteString ("1: Get the record.");
 GotoXY ( 9, 16 );
 WriteString ("2: Delete.");

 IF aList [ i ].type = "R" THEN
  GotoXY ( 9, 17 );
  WriteString ("3: Put a record.");
  GotoXY ( 9, 18 );
  WriteString ("4: Insert to right.");
  GotoXY ( 9, 19 );
  WriteString ("5: Insert to left.");
 END;

 GotoXY ( 5, 20 );
 WriteString ("Choice? ");
 Read ( ch );

 IF ( aList [ i ].type # "R" ) AND
    ( ( ORD ( ch ) - ORD ("0") ) > 2 ) THEN
  terminate ( 5, 21,
     "Wrong file type for this operation.");
 ELSE
  CASE ch OF
   "1": getNode ( aList [ i ].pointer,
                  aList [ i ].position, a );
        GotoXY ( 5, 21 );
        WriteString
("The following record was obtained.");
        showRec ( a, 1, 22 );
  | "2": delete ( aList [ i ].pointer,
                  aList [ i ].position, a );
        GotoXY ( 5, 21 );
        WriteString
  ("Deleted record is/was as follows.");
        showRec ( a, 1, 22 );
  | "3": GotoXY ( 5, 21 );
         WriteString
 ("Provide a record to replace existing entry.");
        getRec ( a, 1, 22 );
```

```
            putNode ( a, aList [ i ].pointer,
                      aList [ i ].position );
      | "4": GotoXY ( 5, 21 );
             WriteString
     ("Provide a record to insert to right of •••
                 •••existing entry.");
            getRec ( a, 1, 22 );
            insertRight ( a, aList [ i ].pointer,
                          aList [ i ].position );
      | "5": GotoXY ( 5, 21 );
             WriteString
     ("Provide a record to insert to left of •••
                 •••existing entry.");
            getRec ( a, 1, 22 );
            insertLeft ( a, aList [ i ].pointer,
                         aList [ i ].position );
      ELSE ;
       (* loop again until input fits the choices *)
      END;(*case*)
     END;(*inner if then else*)
    END;(*if*)

    GotoXY ( 0, 0 );
    WriteString ("Exit <manipulate>? ");
    Read ( ch );

    IF ( ch = "y" ) OR ( ch = "Y" ) THEN
     EXIT;
    END;
   END;(*loop*)
END manipulate;

PROCEDURE mainMenu () : CARDINAL;

VAR choice : CHAR;

BEGIN
 LOOP
   ClearScreen;
   listLists;
   GotoXY ( 13,  3 );
   WriteString
      ("Test ListOps Module/Main Menu");
   GotoXY ( 13,  4 );
   WriteString
  ("============================");
   GotoXY ( 15,  6 );
   WriteString ("0: Terminate program");
   GotoXY ( 15,  8 );
   WriteString ("1: Initialize a List");
   GotoXY ( 15, 10 );
   WriteString ("2: Destroy a List");
   GotoXY ( 15, 12 );
   WriteString ("3: Merge/Append Lists");
   GotoXY ( 15, 14 );
   WriteString ("4: Show Specs of a List");
   GotoXY ( 15, 16 );
   WriteString ("5: Manipulate a List");
   GotoXY ( 15, 18 );
   WriteString ("6: Add an item");
   GotoXY ( 15, 20 );
   WriteString ("7: Remove an item");
   GotoXY (  5, 22 );
   WriteString ("Enter choice: ");
   Read ( choice );

   IF ( choice <= "7" ) AND
      ( choice >= "0" ) THEN
    EXIT;
   END;
  END;(*loop*)

  RETURN ( ORD ( choice ) - ORD ("0") );
END mainMenu;

BEGIN
 FOR i := 0 TO 9 DO
  aList [ i ].type := 0C;
  setListToNil ( aList [ i ].pointer );
  setPositionToNil ( aList [ i ].position );
 END;

 LOOP
  CASE mainMenu () OF
   0 : EXIT;
  | 1 : newList;
  | 2 : giveUpList;
  | 3 : mergeAppend;
  | 4 : showSpecs;
  | 5 : manipulate;
  | 6 : addItem;
  | 7 : removeItem;
  ELSE (* nothing *) ;
  END; (*case*)
 END; (*loop*)

 FOR i := 0 TO 9 DO
  IF NOT nilList ( aList [ i ].pointer ) THEN
   freeList ( aList [ i ].pointer, item );
  END;
 END;
END TestListOps.
```

USUS
P.O BOX 1148
LA JOLLA, CA 92038

ADDRESS CORRECTION REQUESTED

FIRST CLASS MAIL

March 1989

USUS NewsLetter

Volume 3
William D. Smith, Editor Number 3

NewsLetter Publication Dates

NewsLetter	Due date Code/Forms	Due date Articles	Due date Short stuff
May/June 89	04/15/89	04/28/89	05/05/89
July/Aug 89	06/15/89	06/30/89	07/07/89
Sept/Oct 89	08/15/89	08/25/89	09/01/89
November 89	10/01/89	10/13/89	10/20/89

Next NewsLetter coming April