US NewsLetter

Serving the Pascal, Modula-2, and Portable Programming Community Vol. 4 No. 3 May / Jun 1990

IN THIS ISSUE

The USUS NewsLetter is published 6 times per year by USUS, the UCSD Pascal System User's Society, P.O. Box 1148 La Jolla, California 92038. The NewsLetter is a direct benefit of membership in USUS.

Tom Cattrall Editor
Robert Geeslin Publisher

The Formal U.S. Response to ISO on the Draft Proposal for Modula-2

The IEEE working group charged with representing the interests of the United States to WG13 of SC22 of JTC1 of the ISO/IEC requests that the United States register its vote against the publication of DP10514 (SC22/N738, referred to here as D106) as a Draft International Standard for the following reasons:

0. We believe that the US community would not accept the current Draft Proposal (DP) because it neither codifies established practice, nor gives adequate justification for its differences therefrom. In 1987 the U.S. committee drafted this statement.

The general philosophy of the committee and what the committee felt to be the charter for Modula-2 standardization:

1) Clarify imprecisions and contradictions in the language as we know it
2) Avoid removing language features unless necessary
3) Avoid changing language features without good reason
4) Avoid semantic changes that are not associated with syntactic changes
5) Minimize language extensions
6) 'Deprecate' obsolete features rather than removing them (i.e., have compilers accept these features while issuing warning messages, and warn users that such features may not be supported in the long term)
7) Flag dangerous programming practices, often by importing from SYSTEM.

We are disappointed that WG13 has been unable to state its goals or philosophy.

1. The document is incomplete, has many open 'to-do' items, small errors, etc. We are extremely concerned that there has been no formal (e.g. automated) verification of the VDM.

2. Although the definition of Modula-2 should be fully and separately supported both by VDM and by English text, we find that in the D106 neither method adequately defines the language. D106 uses English text too sparingly and the text is often incomplete and inadequate. The dialect of VDM used in D106 is still undefined, leaving the DP in the

illogical position of providing an undefined definition of Modula-2. We cannot support the definition given by the D106 since we cannot know what the definition means. A reference to an out-of-print book [Jones80] is not a legitimate substitute for a definition, especially as that book does not describe the dialect of the VDM used in D106; neither is a reference to a moving target such as the new VDM-SL NWI. We agree that having the VDM is a good idea, provided the formalism has a complete, stable, and publicly available definition, either included in the DP or referenced by it. If not so defined, the formalism must be removed.

3. Alternative tokens Sec.5.5. We still hold our position of August 1988 that the alternative symbols for square brackets ("(.", ".)") create unnecessary syntactic ambiguity to no positive gain. We reiterate our proposal that "(!" and "!)" be used instead.

4. Requirements clauses Sec.4.11. The various minima suggested in Sec.4.11 are unnecessary, technically problematical, and have the potential of delaying the standards process if adopted. We propose that all suggested minima be eliminated except for the single requirement that SET be large enough to cover CHAR.

5. Value constructors Sec.6.7.5. There is general consensus against array and record non-constant value constructors. They add no new functionality at significant cost both to the definition and to compiler complexity. Many would be willing to undo Wirth's dynamic set value constructors, if that were necessary to get rid of array and record value constructors.

6. Comment bodies Sec.5.8.1. We believe the intent of the language comment facility is

a. Comments shall not change the meaning of the program, and
b. As far as possible, one should be able to convert any piece of legal program text into comment by enclosing it in "(* ... *)" brackets, regardless of the presence in that text of comments, compiler directives, strings, or other constructs.

It is important that the DP clearly state this intent, even if it cannot properly be captured in a syntax specification, and that the implementor be encouraged to honor this intent.

7. Compiler Directives Sec.5.8.2. We note several problems with compiler directives:

a. A "$" is a national currency symbol, and is not therefore appropriate as the default directive symbol.
b. Arbitrary white space before the "$" unnecessarily complicates scanning the directive.
c. The nesting interaction between directives and normal comments is likely to cause problems.

We reiterate our suggestion of 1988 that "<*" and "*>" be used as bracketing symbols. It should also be stated that compiler directives can be commented out.

8. Machine Addresses in Variable Declarations Sec.6.2.5. We request that MACHINEADDRESS be a record type in accordance with the proposal presented at Linz by Keith Hopper of New Zealand. In D106, MACHINEADDRESS is a function; since function calls cannot appear in constant expressions, this precludes the intended use of MACHINEADDRESS as a way to specify the address of a variable.

9. Constant Expressions Sec.6.7.7. This section requires constant real expressions to yield the same result whether computed at compile time or at run time. Although we note that a significant subset of the potential users of Modula-2 insist on this property, it causes a severe problem on machines where real arithmetic is dynamically changeable. For example, on a machine with IEEE-754 floating point, the declaration,

```
CONST HalfPi = PI / 2.0;
```

will compute a value dependent on the current rounding mode of the floating-point operations. Since this mode can be changed during program execution, the computed value cannot in general be decided at compile time. While this example could in principle be dealt with by deferring evaluation of the constant until execution time, the problem is insuperable in cases such as

```
TYPE A = ARRAY[1..TRUNC(expr)] OF INTEGER;
```

where the value is required at compile time but cannot then be computed. This problem should be addressed and resolved.

10. Exceptions Sec.6.12, Sec.7.3, Sec.8.2, Annex G. D106 does not include a single complete model for adding an exception-handling capability to Modula-2. Since most of the proposed libraries presuppose that such a capability is defined, the failure to reach closure on this issue has an impact on both the definition and the standardization process. The evaluation criteria offered in N328 Sec.3 appear sound to us, and in the light of these criteria we observe

a. Any proposal meeting these criteria is likely to involve a language change too great to be admissible at this stage in the standardization process, and

b. Any proposal not involving a language change is unlikely to offer sufficient benefit to justify its inclusion.

Accordingly, we propose that all references to exception modules be removed from the DP. If WG13 believes that one final attempt should be made to revise this facility, then we suggest that a very simple and primitive setjmp/longjmp model, such as D70/N247 proposal 2, if not overly embellished, may provide an approach that can yield an acceptable minimal proposal in the time available.

11. CAST <> bitsize Sec.7.1.3.4. The definition of CAST should be changed so that:

a. The effect of CAST on values of ambiguous sizes (i.e.the abstract types Z, R) is implementation-dependent,
b. The CAST of a typed value into a type of different size is implementation-dependent, and
c. A CAST of a value designator whose alignment is "incorrect" for the target type is implementation-dependent.

12. Lexis of Identifiers Sec.5.3. If low-line ("_") is to be permitted as a component of an identifier, then it should be permitted wherever a letter is permitted. The lexis should not be made more complicated merely to enforce one body's view of good taste.

13. Termination Sec.7.1.2. We reiterate the US position of 1989 as described in P155, dated 4 August 89. Termination as drafted is unnecessarily complex, provides little additional benefit over P155, and has undesirable interactions with exception handling and coroutines.

14. Forward reference pointer types. The problem exhibited by the following apparently legal code fragment should be resolved:

```
TYPE T=Q;
PROCEDURE a;
  VAR v: ^T;
  PROCEDURE b;
  BEGIN
    v^:=x; (* what code generated by one pass
              compiler? *)
  END b;
  TYPE T=R;
END a;
```

15. SYSTEM module Sec.7.1. In abstracting a storage model underlying SYSTEM, the D106 lets the abstract model pervade the definition module. Thus what should have been simple access to machine primitives has become complex operations on pieces of storage that are rarely directly supported by the native hardware. The DP should not require module SYSTEM to export constant and type identifiers other than the following:

```
CONST
  LOCSPERWORD = (* implementation defined *);
  BITSPERLOC = (* implementation defined *);
  BITSPERWORD = BITSPERLOC*LOCSPERWORD;
TYPE
  LOC; (* an opaque type equivalent to SET OF
          [0..BITSPERLOC-1] *)
  WORD; (* an opaque type equivalent to SET
           OF [0..BITSPERWORD-1] *)
  BITSET = SET OF [0..BITSPERWORD-1];
  ADDRESS = POINTER TO LOC;
  MACHINEADDRESS = RECORD
                     (* implementation
                        defined *)
                   END;(* required by KH pro-
                          posal *)
```

The identifier ADDRESSVALUE should be removed, as CAST is sufficient. Furthermore, the system function procedures SHIFT and ROTATE should operate on and return values of type WORD so defined:

```
PROCEDURE SHIFT  (value: WORD;
                  amount: INTEGER): WORD;
PROCEDURE ROTATE (value: WORD;
                  amount: INTEGER): WORD;
```

As decided by WG13, we request that MACHINEADDRESS be a record type to be used in fixing a hardware address value in the manner proposed by Keith Hopper (see #8 above) as follows:

```
VAR v [MACHINEADDRESS{MediumModel,0FEH,0D00DH}]:
      INTEGER;
```

16. String constant catenation Sec.5.5.2.5. A different symbol is needed for string literal catenation, as "||" causes an ambiguity with case list separators; we suggest overloading the binary operator "+". There is also a problem with using the null string to denote the string termination character. The expression "'a'+''+'b'" should be the same as "'ab'" and not insert a string terminator in the middle.

17. Type transfer Annex E, p.16 (see CAST Sec.7.1.3.4). We were unable to find in D106 the type transfer of PIM (Programming in Modula-2 by Wirth). WG13 agreed to retain this feature but deprecate it. We suggest that, similar to NEW and DISPOSE, the old form of type transfer might require that SYSTEM.CAST be visible.

18. INTEGER and CARDINAL Sec.6.9.1. We were unable to find in D106 the relationship between the ranges of INTEGER and CARDINAL and the bounding constants known as K1, K2, and K3.

19. WG13 agreed that all changes from PIM would be noted in the document, and we observe that many are. We require that all changes from and clarifications to PIM be noted. We also request that in each future draft, all additions, changes, and deletions from the previous draft be marked with change bars or other appropriate mechanism, to facilitate proper and efficient evaluation of the draft proposals.

20. Coroutines Sec.7.2. While we can support moving coroutines to a separate module, we cannot support the syntactic and semantic changes made from PIM as they add no functionality and break existing code.

21. I/O Library Sec.9.2. Noting the lack of goals or rationale, as well as the size, complexity, and novelty of the proposed I/O library, we cannot support the proposal in D106. WG13 requested a small and simple I/O library be produced. We reiterate that request. We note that D75 of August 1988, which tried to address that request, has not been allowed to mature.

22. Strings Module Sec.9.4. The Strings module uses the look-ahead philosophy that WG13 purged from the I/O library after much debate. The current module is a radical departure from all previous implementations, complicates code, and is of no clear benefit to the programmer.

23. Module protection Sec.6.1.11; Procedure protection Annex G. Module protection, priority, and associated syntax and semantics should be removed from the Draft Proposal because:

a. They are an extension to the language defined in PIM,
b. They assume a non-universal machine model,
c. They assume particular process and synchronization models, and
d. They will cause serious problems when multi-processor Modula-2 is designed.

24. LowReal and LowLong Sec.8.3.1, Sec.8.3.2. The modules LowReal and LowLong currently specify a set of constants that define various parameters of the floating-point implementation. These values become invalid if SetMode() is ever called. A much more usable definition would be:

```
TYPE
   FPInfoValues = (FPIEEE, FPISO, FPRounds,
                   FPGUnderflow, FPException);
   FPInfo = SET OF FPInfoValues;

   PROCEDURE GetFPInfo(): FPInfo;
     (* returned value is valid until next call of
        SetMode() *)
     (* Comment: may want to expand FPException *)
```

25. Concurrent Programming Modules Sec.9.3. The D106 has significant flaws:

a. It mixes event handling with basic process operations.
b. Important operations on process queues are not in this module, namely, Lock, EnterQueue, SuspendMe, and a way to release a lock.
c. In the Semaphores module the definition is for general semaphores, not counting semaphores (the two are distinct), and the operator set is incomplete (e.g. no indivisible VP()).

Summary of Balloting on the Modula-2 Draft Proposal

The summary of the ballot is as follows:

Members supporting the proposal without comments: 4
Belgium, Finland, Hungary, Italy

Members supporting the proposal with comments: 3
Canada, Denmark, Japan

Members not supporting the proposal: 5
France, Germany FR, Netherlands, UK, USA

Members not voting: 8
Of these, China and Switzerland have had participation
and so has New Zealand (and we know they tried to vote!)

The fifth meeting of SC22/WG13 is to be held from Monday June 4th to Friday June 9th 1990 in Milton Keynes, UK. The main business of the meeting is to review and process comments received on the Modula-2 draft proposal, to produce a response in the form of a disposition of comments paper, and to reach decisions on outstanding issues affecting the production of the next draft of the Standard.

Discussion of a Standard Modula-2 String Module

An extract from the formal response to ISO on the Draft Proposal for Modula-2 criticizes the D106 draft as follows:

22. Strings Module Sec.9.4. The Strings module uses the look-ahead philosophy that WG13 purged from the I/O library after much debate. The current module is a radical departure from all previous implementations, complicates code, and is of no clear benefit to the programmer.

The purpose of this note is to amplify on that extract.

We note that there seems to be remarkable consensus among existing libraries as to what should be in a Strings module, and what the parameters to various routines should be.

In keeping with our belief that the Standard should reflect existing practice wherever possible, the US requests that the standard specify a Strings module that can be summarized most simply as follows, noting that the semantics of these routines are essentially as in the corresponding places in D106.

```
DEFINITION MODULE Strings;

PROCEDURE Length (StringValue : ARRAY OF CHAR) : CARDINAL;
(* returns the length of StringValue *)

PROCEDURE Assign (Source : ARRAY OF CHAR; VAR Destination : ARRAY OF CHAR);
(* copies Source to Destination *)

PROCEDURE Extract (Source : ARRAY OF CHAR;
                   StartIndex,  NumberToExtract : CARDINAL;
                   VAR Destination : ARRAY OF CHAR);
(* copy NumbertoExtract characters from Source to
   Destination, starting at StartIndex in Source *)

PROCEDURE Delete (VAR StringValue : ARRAY OF CHAR;
                  StartIndex, NumberToDelete : CARDINAL);
(* delete NumberToDelete chars from StringValue,
   starting at position StartIndex *)

PROCEDURE Insert (Source : ARRAY OF CHAR; VAR Destination : ARRAY OF CHAR;
                  StartIndex : CARDINAL);
(* insert Source into Destination at position StartIndex *)

PROCEDURE Concat (StringVal1, StringVal2 : ARRAY OF CHAR;
                  VAR Destination : ARRAY OF CHAR);
(* concatenate StringVal2 to StringVal1 and place the resulting string
   in Destination *)


TYPE
  CompareResult = (less, equal, greater);

PROCEDURE Compare (StringVal1,StringVal2 : ARRAY OF CHAR) :  CompareResult;
(* Return: less, equal, greater according as StringValue1 < = > StringValue2 *)

PROCEDURE FindNext (Pattern, StringValue : ARRAY OF CHAR;
                    StartIndex : CARDINAL; VAR PatternFound : BOOLEAN;
                    VAR PosOfPattern : CARDINAL);
(* look for next occurrence of Pattern in StringValue, starting search from
   StartIndex.  PosOfPattern returns start position of Pattern if PatternFound *)

END Strings.
```

Notes:

After examining about 18 libraries or library proposals we make the following observations:

`PROCEDURE Length -`

There is unanimity among vendors about this one. D106 introduces the pervasive LENGTH in its place.

`PROCEDURE Compare -`

10 vendors use INTEGER, 4 use an enumeration type as the result type. We regard the adoption by D106 of the enumeration as preferable.

`PROCEDURE Extract -`

Many vendors call this "Copy" or "SubStr". The D106 name seems a good compromise, noting the misleading interpretation that could be placed on "Copy".

`PROCEDURE Insert -`

Nearly all libraries examined had parameters in this order. D106 was in a minority of 1 in deviating (presumably in an attempt to "tidy up").

`PROCEDURE Concat -`

D106 does not have this one at all. 14 of the libraries examined had it, and 12 have the parameters in this order. It seems very popular. Those few who do not have Concat have Append, and justify this by saying that most Concat operations are really Append. However Append is an easy call of Concat, and the reverse is not true. Furthermore, of the 6 libraries examined that supplied Append, most had the parameters in the opposite order to those given in D106. Finally we note that

the absence of Concat has nothing to do with the proposed concatenation operator for string literals.

```
PROCEDURE FindNext -
```

The popular alternative is something like

```
PROCEDURE Pos (Pattern,
               StringValue : ARRAY OF CHAR) :
               CARDINAL;
(* Returns  index of first occurrence of Pattern in
  StringValue or > HIGH(StringValue) if no match is
  found *)
```

Use of this is, admittedly, awkward. 9 vendors have it like this. 4 more discerning ones have

```
PROCEDURE Pos (Pattern, StringValue : ARRAY OF CHAR;
               StartIndex : CARDINAL) : CARDINAL;
```

and the other three have other ways of returning the "result". The idea that 0 ... Length(StringValue)-1 implies Success and >HIGH(S) (or variations on this theme) implies failure is awkward. The D106 idea has clear merit. Traditional Pos is easily written in terms of FindNext, but the reverse is not true.

Summary

• We request the substitution of Concat for Append.

• We request the retention of Length, and that consideration be given to the deletion of the pervasive LENGTH.

• Insert should have parameters to confirm to existing practice.

• FindPos is an improvement over the "popular" Pos, and could be retained as the old Pos is easily defined in terms of it.

• We request the deletion of the following. If WG13 is keen to standardize them we ask that they be placed in a subsidiary library.

String1
BigStringCapacity
BigString
CanAssignAll
CanExtractAll
CanDeleteAll
CanInsertAll
CanReplaceAll
Replace
CanAppendAll
Append
Capitalize
FindPrev
FindDiff

Error detection

We note that there are 5 approaches that could be taken to run time error handling in a strings library

(a) Ignore them - the D106 procedures never fail to return some sort of result (though it may not be what one intuitively wishes). This seems to be the common practice, and is in line with the proposal above.

(b) Provide pre-event predicates (as in D106) so that conscientious programmers could test, and the more daring or traditional could follow prevailing practice at their peril. Academically this has some merit over all the others, but we doubt that it is necessary to specify it in full detail, as the predicate functions are easy to write, even if the various Insert, Concat, Find, etc. ones are tricky because of the awkward Modula string representation.

(c) Provide the module with "state", and a post event predicate

```
PROCEDURE LastStringResult () : SomeEnumerationType;
```

to allow conscientious programmers to pick up the mess later. We have never seen this anywhere, but the analogies with various libraries that take this approach to I/O are obvious.

(d) Provide each routine with an extra VAR OK : BOOLEAN parameter to force the programmer to be aware of error possibilities, even if he does not bother to check them. This approach was advocated by the Ad-Hoc Library in 1984, but we have not seen it put into many current libraries.

Note that "post testing" for string using either of (c) or (d) is "safe", as even in the cases of failure one gets some result, and does not crash the system (as one might do when dividing by zero etc).

(e) Raise an exception. This does not appear to be common practice (with present implementations "raise exception" would almost always mean "run time error and halt").

Some Ideas about Pascal Programming Nomenclature

David Craig
736 Edgewater, Wichita, Kansas 67230
March 1990

In my 10 years programming with Pascal for various machines I've developed a simple set of nomenclature guidelines for Pascal programs. The nomenclature guidelines are as follows:

Pascal reserved words;

- All reserved words are in uppercase characters
  Example: PROGRAM, UNIT, BEGIN, END, CONST, TYPE, VAR, ARRAY

Program procedure and function names;

- Application routine names start with a lowercase character, contain underscores between routine name words, and the routine name words start with uppercase characters.
  Example: find_Expression_Token, list_DataBase_File_Header

- System or built-in library routine names start with an uppercase character.
  Example: GetNextEvent, ZoomWindow

- All local routine constants, types, and variables are in lowercase.
  Example: time_of_day, list_index, user_is_done

- All local routine constant names start with the prefix 'k_'
  Example: k_size_of_list

- All local routine type names start with the prefix 't_'
  Example: t_paoc, t_current_window_ptr

- Underscore characters should be used as much as possible in all names since this character improves the readability tremendously.
  Example: use time_of_day instead of timeofday

Program globals;

- Global constants start with the prefix 'gc_' (Global Constant)

- Global types start with the prefix 'gt_' (Global Type)

- Global variables start with the prefix 'gv_' (Global Variable)
  Example: gc_List_Size, gt_PAOC, gv_Application_Parms

- Global types and global variables based upon those types should have similar names
  Example: If the type is named gt_MyType, then the variable should be named gv_MyType

Unit interface routines;

- Unit routine names start with a two character prefix that corresponds to the unit name
  Example: EP_Tokenize_Expression for a routine in a unit named Expression_Parser

- Unit global constants start with the unit name prefix and 'c_'

- Unit global types start with the unit name prefix and 't_'

- Unit global variables start with the unit name prefix and 'v_'
  Example: EP_c_Max_Token_Length , EP_t_Token_List, EP_v_Token_Offset

Unit implementation globals;

- Global constants start with the prefix 'pc_' (Private Constant)

- Global types start with the prefix 'pt_' (Private Type)

- Global variables start with the prefix 'pv_' (Private Variable)
  Example: pc_Max_Name_Length, pt_Dynamic_List, pv_Unit_Init_State

These nomenclature ideas have proved their worth in many of my Pascal programs. For small programs, they may not be too useful, but for large programs (5,000+ lines) that require several months to write these guidelines are to me almost indispensable. Other programmers may consider these ideas a waste of time. But the small waste of time it takes to use these ideas, in my opinion, will be far less than the waste of time at a future date when a programmer is trying to understand either his or another person's program which does not follow these guidelines.

These ideas may be difficult to implement with compilers that handle only short names. For example, my Lisa Pascal compiler only uses the first 8 characters of names while my Macintosh MPW Pascal compiler uses the first 63 characters.

These ideas are based loosely upon the "Hungarian" style that MicroSoft uses for its C projects. Any comments, positive or negative, about these ideas are welcome. The ultimate goal of these ideas is to make programming easier to understand and maintain.

Data Entry / Validation Module

Peter M. Perchansky
211 South 5th Street
Womelsdorf, PA 19567

PMPData is a data entry/validation library module for Modula-2 written in TopSpeed Modula-2, version 1.17.

The main procedure in PMPData is UserInput. The code in UserInput allows you to make one procedure call to perform data entry & editing of most data types (standard plus string), in addition to calling client-module defined validation procedures.

The ability to provide entry and editing on most data types is handled through the use of generics.

Data is passed to and from UserInput in the form of a stream (array) of bytes. This allows any data type to be passed to UserInput, even though UserInput only handles standard data types and strings.

Data validation is provided by passing the address of the input data to the validation procedure. This allows UserInput to pass one address to one validation procedure since a variable of type address can store any data type.

Types used in the library:

```
dataType = (boolean, cardinal, char, integer,
            real, longcard, longint, longreal,
            stringArray);
```

Types of dataType define what (standard) type of data is being used for data input or validation. dataType is an enumeration consisting of all the standard data types in addition to the string (ARRAY OF CHAR) data type.

```
validationProc = PROCEDURE (ADDRESS, dataType):
                 BOOLEAN;
```

Types of validationProc are procedure variables that will be called to test the validity of data input. The procedure variable consists of an address for the data being tested, and the type of data being tested. The procedure variable should return TRUE if the input data passed the test or FALSE if the input data is invalid.

Basic algorithm used for the UserInput Procedure:

```
PROCEDURE UserInput (prompt: ARRAY OF CHAR;
                     VAR data: ARRAY OF BYTE;
                     length: CARDINAL;
                     type: dataType;
                     validate: validationProc);
```

A prompt (can be an empty string), a VAR'd data variable, a desired input length, a data type, and a validation procedure variable are passed to the UserInput Procedure.

If the prompt is not an empty string, then it is written to standard output (usually the screen).

A temporary input buffer (string) is used to read character by character from standard input (usually the keyboard) until the enter (return) key is pressed; full editing functions are allowed (home, end, insert, escape, delete, back space, left and right arrow keys).

Only length number of characters are allowed to be entered by the user.

Once the enter (return) key is pressed, the input buffer (string) will be converted to the appropriate data type, and moved into the VAR'd data variable.

Then the validation procedure variable will be called. The NoValidation Procedure (dummy procedure that always returns TRUE) can be called if no validation is necessary. Otherwise a client-module defined procedure is used.

If the data does not pass the validation tests of the validation procedure, then the user must re-enter the data until valid data is entered.

Sample calls:

```
UserInput ("Enter your name:  ", name, 25,
           stringArray, NameCheck);

UserInput ("Enter your age:  ", age, 2, cardinal,
           NumberCheck);

UserInput ("Are you married?  ", married, 1,
           boolean, NoValidation);

UserInput ("Enter your score:  ", testScore, 5,
           real, NumberCheck);
```

A WrLn should be called after every UserInput, if you want the entries on separate lines.

Sample validation procedures:

```
PROCEDURE NameCheck (nameAddr: ADDRESS; type: dataType);
    VAR
        x, y : CARDINAL;
    BEGIN
        IF CHAR (nameAddr^) # CHR (0) THEN
            RETURN TRUE
        ELSE
           ReadCursorPos (x, y);
           GotoXY (25, 20);
           WrStr ("You must enter a name.");
           WrChar (CHR (7));
           GotoXY (x, y);
           RETURN FALSE;
        END;
    END NameCheck;



PROCEDURE NumberCheck (numberAddr: ADDRESS; type: dataType);
    VAR
        x, y : CARDINAL;
        ok   : BOOLEAN;

    BEGIN
        CASE type OF
           |  cardinal : ok := CARDINAL (numberAddr^) > 0
           |  longcard : ok := LONGCARD (numberAddr^) > 0
           |  integer  : ok := INTEGER (numberAddr^) > 0
           |  longint  : ok := LONGINT (numberAddr^) > 0
           |  real     : ok := REAL (numberAddr^) > 0.0
           |  longreal : ok := LONGREAL (numberAddr^) > 0.0
        END;

        IF ok THEN
           RETURN TRUE
        ELSE
           ReadCursorPos (x, y);
           GotoXY (25, 20);
           WrStr ("Number must be greater than zero.");
           WrChar (CHR (7));
           GotoXY (x, y);
           RETURN FALSE;
        END;
    END NumberCheck;
```

Please feel free to contact me if you have any questions or comments.

See the following pages for the listings of the PMPData module.

```
(*=============================================================
  =  Program Id:      PMPDATA.DEF                            =
  =  Programmer:      Peter M. Perchansky                    =
  =  Purpose:         Export data entry & validation procedures.  =
  =  Date Written:    04-15-90                               =
  =-----------------------------------------------------------=
  =                      Version History                      =
  =-----------------------------------------------------------=
  =  04-15-90  PMP  Date written.                            =
  =============================================================*)

DEFINITION MODULE PMPData;

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


TYPE
    dataType       = (boolean, cardinal, char, integer, real,
                      longcard, longint, longreal, stringArray);
    validationProc = PROCEDURE (ADDRESS, dataType): BOOLEAN;


PROCEDURE NoValidation (data: ADDRESS; type: dataType): BOOLEAN;
(* User can call this procedure if no data validation is necessary.   *)

PROCEDURE UserInput (prompt: ARRAY OF CHAR; VAR data: ARRAY OF BYTE;
                     length: CARDINAL; type: dataType;
                     Validated: validationProc);

(* Prompt user for data (if prompt is not blank) of specified length  *)
(* from standard input.  User must press enter (return) for the data  *)
(* to be accepted.  type specifies the actual data type of the input. *)
(* Validated is a client-module defined procedure used to validate    *)
(* the user's input.  UserInput will not return until valid input     *)
(* is entered by the user.                                            *)

END PMPData.


(*=============================================================
  =  Program Id:      PMPDATA.MOD                            =
  =  Programmer:      Peter M. Perchansky                    =
  =  Purpose:         Export data entry & validation procedures.  =
  =  Date Written:    04-15-90                               =
  =-----------------------------------------------------------=
  =                      Version History                      =
  =-----------------------------------------------------------=
  =  04-15-90  PMP  Date written.                            =
  =============================================================*)

IMPLEMENTATION MODULE PMPData;

(*-------------- Procedures from JPI's TopSpeed Modula II -------------*)
FROM AsmLib    IMPORT Move;
FROM IO        IMPORT RdCharDirect, WrChar, WrCharRep, WrStr;
FROM Str       IMPORT Insert, Delete;
FROM SYSTEM    IMPORT ADDRESS, ADR, BYTE;
(*---------------------------------------------------------------------*)

(*-------------- Procedures from PMP's Library Modules ----------------*)
FROM PMPChar   IMPORT FirstChar;
FROM PMPConv   IMPORT StrBool, StrCard, StrInt, StrReal, StrLongCard,
                      StrLongInt, StrLongReal;
FROM PMPDos    IMPORT GotoXY, ReadCursorPos;
FROM PMPRange  IMPORT MinCard;
FROM PMPStr    IMPORT LocateEnd, InitStr, EmptyStr;
(*---------------------------------------------------------------------*)

(*---------------------------------------------------------------------*)
(*       Global constants/variables used internally by PMPData.        *)
(*---------------------------------------------------------------------*)

CONST
    null           = CHR (0);
    home           = CHR (1);
    rightArrow     = CHR (4);
    end            = CHR (6);
    delete         = CHR (7);
    backSpace      = CHR (8);
    return         = CHR (13);
    leftArrow      = CHR (19);
    ins            = CHR (22);
    escape         = CHR (27);


(*---------------------------------------------------------------------*)
(*          Procedures exported by PMPData.                            *)
(*---------------------------------------------------------------------*)

PROCEDURE NoValidation (data: ADDRESS; type: dataType): BOOLEAN;
(* User can call this procedure if no data validation is necessary.   *)

BEGIN
    RETURN TRUE;
END NoValidation;

PROCEDURE UserInput (prompt: ARRAY OF CHAR; VAR data: ARRAY OF BYTE;
                     length: CARDINAL; type: dataType;
                     Validated: validationProc);

(* Prompt user for data (if prompt is not blank) of specified length  *)
(* from standard input.  User must press enter (return) for the data  *)
(* to be accepted.  type specifies the actual data type of the input. *)
(* Validated is a client-module defined procedure used to validate    *)
(* the user's input.  UserInput will not return until valid input     *)
(* is entered by the user.                                            *)

VAR
    dataAddr  : ADDRESS;                (* address of converted data  *)
    dataSize,                           (* size of data holder        *)
    x, y, i   : CARDINAL;               (* positional variables       *)
    insert    : BOOLEAN;                (* insert state for editing   *)
    inChar    : CHAR;                   (* character input variable   *)
    buffer    : ARRAY [0..80] OF CHAR;  (* temporary buffer for input *)
```

```
    tBool     : BOOLEAN;                (* temporary boolean          *)
    tCard     : CARDINAL;               (* temporary cardinal         *)
    tChar     : CHAR;                   (* temporary char             *)
    tInt      : INTEGER;                (* temporary integer          *)
    tReal     : REAL;                   (* temporary real             *)
    tLongcard : LONGCARD;               (* temporary long cardinal    *)
    tLongint  : LONGINT;                (* temporary long integer     *)
    tLongreal : LONGREAL;               (* temporary long real        *)

BEGIN
    IF NOT EmptyStr (prompt) THEN
       WrStr (prompt)
    END;

    IF type = stringArray THEN
       dataSize := MinCard (HIGH (data), HIGH (buffer)) + 1;
       IF length > dataSize THEN
          length := dataSize
       END;
    END;

    InitStr (buffer);                   (* initialize buffer to null    *)
    insert := TRUE;                     (* start with insert state TRUE *)
    ReadCursorPos (x, y);               (* save current cursor location *)
    WrCharRep (' ', length);            (* clean up the input line      *)
    GotoXY (x, y);                      (* move to saved location       *)

    REPEAT
         i := 0;
         LOOP
              IF i > length THEN
                 i := length;
                 buffer[i] := null;
              END;

              WrStr (buffer);
              IF LocateEnd (buffer) < length THEN
                 WrChar (' ')           (* keep one char ahead clear    *)
              END;

              GotoXY (x, y + i);
              inChar := RdCharDirect ();
              IF inChar = null THEN     (* extended character set       *)
                 CASE RdCharDirect () OF
                    | CHR (71) : inChar := home
                    | CHR (75) : inChar := leftArrow
                    | CHR (77) : inChar := rightArrow
                    | CHR (79) : inChar := end
                    | CHR (82) : inChar := ins
                    | CHR (83) : inChar := delete
                 END;
              END;

              CASE inChar OF
                | ' '..'~' : IF insert THEN
                                Insert (buffer, inChar, i)
                             ELSE
                                buffer[i] := inChar
                             END;
                             INC (i);
                | home      : i := 0
                | end       : i := LocateEnd (buffer)
                | leftArrow : IF i > 0 THEN
                                 DEC (i)
                              END;
                | rightArrow: IF i < LocateEnd (buffer) THEN
                                 INC (i)
                              END;
                | delete    : Delete (buffer, i, 1)
                | backSpace : IF i > 0 THEN
                                 DEC (i);
                                 Delete (buffer, i, 1);
                              END;
                | ins       : insert := NOT insert;  (* toggle insert *)
                | escape    : buffer[0] := null;
                              GotoXY (x, y);
                              WrCharRep (' ', length);
                              i := 0;
                | return    : EXIT
              END;

              GotoXY (x, y);
         END;

         (* NOTE: ADR (StrCard (buffer)) <as well as others> does     *)
         (*       not yield the same results as those presented below *)

         CASE type OF
           | boolean     : tBool := StrBool (buffer);
                           dataAddr := ADR (tBool);
           | cardinal    : tCard := StrCard (buffer);
                           dataAddr := ADR (tCard);
           | char        : tChar := FirstChar (buffer);
                           dataAddr := ADR (tChar);
           | integer     : tInt := StrInt (buffer);
                           dataAddr := ADR (tInt);
           | real        : tReal := StrReal (buffer);
                           dataAddr := ADR (tReal);
           | longcard    : tLongcard := StrLongCard (buffer);
                           dataAddr := ADR (tLongcard);
           | longint     : tLongint := StrLongInt (buffer);
                           dataAddr := ADR (tLongint);
           | longreal    : tLongreal := StrLongReal (buffer);
                           dataAddr := ADR (tLongreal);
           | stringArray : dataAddr := ADR (buffer);
         END;

         Move (dataAddr, ADR (data), length);

    UNTIL Validated (dataAddr, type);   (* call validation procedure  *)
END UserInput;

END PMPData.
```

Dynamic Strings in Modula-2

by Peter M. Perchansky

Modula-2 and several other high level languages force you to declare the size of arrays prior to compilation time.

This can be very frustrating at times because you usually get caught allocating too much space or too little for the array.

Your program should still work if you allocate too much space, but then you are wasting memory. If you allocate too little space... well we should all know what user's of your program will be swearing ;-)

Is there an easier/better way?

Well, start thinking of what exactly is an array.

An array is a continuous block of memory going from a lower bounds to an upper bounds. The block is usually broken down by segments (elements); each segment accessible by some type of index.

Eg.

```
stringArray = ARRAY [0..79] OF CHAR;
```

stringArray (depending on the system) takes up one 80-byte block. There are 80 segments (elements) of one byte each. Each segment is accessible by its corresponding (CARDINAL) index (0..79).

What if you were to do your own assigning of memory blocks? What have you created?

A dynamic array. One that only uses whatever memory is necessary to store the elements contained therein.

You might make the following statement in order to create the above array (dynamic style):

```
ALLOCATE (stringArray, 80);
```

to allocate 80 bytes of memory to the variable stringArray. What TYPE would you make stringArray?

You have at least three choices:

```
stringArray = ADDRESS;
stringArray = POINTER TO BYTE;
stringArray = POINTER TO CHAR;
```

I chose to use ADDRESS as my definition of a dynamic string variable since ADDRESS is more portable than BYTE (not all M2 implementations have access to the BYTE data type); while POINTER TO CHAR was a good choice, it is more misleading to the eye since you would think you where pointing only to one CHARacter. So you allocated 80 continuous blocks of memory. How do you go about accessing each segment of it?

How about a picture?

P	e	t	e	r		M	.		P	e	r	c	h	a	n	s	k	y
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Accessing the first segment is as simple as CHAR (stringArray^); But how would you access the 5th element?

When you allocated your dynamic array, your first character occurs at the base of the ADDRESS. Each character after the first is a given number of bytes from the base.

If you want to access the 5th element, you must advance your address 5 bytes (in the case of CHAR).

TopSpeed Modula-2's library modules come with procedures for Incrementing and decrementing addresses, along with other address manipulation procedures; which is why you will not see my code for such procedures (why reinvent the wheel?).

After you access the elements you want, you must decrement the address back to the base of the array. Otherwise you will lose data or corrupt the memory block (did I state that this was going to be easy?).

How do you know when you have reached the end of a dynamic array? For my purposes, I chose to use the null character (CHR (0)) to denote the end of a dynamic string array.

To make your life easy <grin>, I am publishing my implementation of a module with various procedures for dealing with the creation, assignment, manipulation, and disposal of dynamic strings.

Feel free to make modifications to this module as you see fit; please let me know of your modifications so that I can test them in my programming environment.

You can contact me by writing to the following address:

Peter M. Perchansky
211 South 5th Street
Womelsdorf, PA 19567

```
(*=============================================================*
 =  Program Id:      PMPDYNA.DEF                              =
 =  Programmer:      Peter M. Perchansky                      =
 =  Copyright:       All rights reserved, Peter M. Perchansky, 1990 =
 =  Purpose:         Export dynamic string types and procedures.    =
 =  Date Written:    03-05-90                                 =
 =-------------------------------------------------------------=
 =                        Version History                    =
 =-------------------------------------------------------------=
 =  04-13-90  PMP  Added FIOWrDynaStr & FIORdDynaStr procedures. =
 =  04-10-90  PMP  Added TrimAll procedure.                     =
 =  04-09-90  PMP  Added TrimLeft and TrimRight Procedures.     =
 =  04-02-90  PMP  Added RdDynaStr procedure.                   =
 =  03-24-90  PMP  Modified SizeOfDyna and SizeOfStatic procedures. =
 =  03-12-90  PMP  Released Copyrighted source code into P.D.   =
 =  03-05-90  PMP  Date written.                                =
 *=============================================================*)

DEFINITION MODULE PMPDyna;

(* A dynamic string is a data type that stores an array of [0..high]  *)
(* of CHARacters (high being at least 1 --> the null character).      *)
(* All dynamic strings are null terminated.                           *)

(*------------- Procedures from JPI's TopSpeed Modula II ------------*)
FROM FIO IMPORT File;
(*-------------------------------------------------------------------*)

TYPE
    DynamicString;                                    (* opaque type *)
    DynaStat     = (empty, full, mismatchedSize, none);
    DynaLoc      = DynamicString;
    ErrorHandler = PROCEDURE (DynaStat, DynaLoc);

    (* ErrorHandler is a client-module installable error handling     *)
    (* procedure.  The procedure is called with the status code, and  *)
    (* the name of the procedure where the error occured.  Until a    *)
    (* client-module error handler is installed, all errors will be   *)
    (* ignored.                                                       *)


PROCEDURE InitDyna (VAR dynaStr: DynamicString);
(* Initialize dynamic string for use.  Should be called prior to any  *)
(* operation on a "new" (ie. never assigned) dynamic string varaiable *)
(* This procedure should not be used on dynamic strings that were     *)
(* assigned (use Dispose instead).                                    *)

PROCEDURE LastError (): DynaStat;
(* Returns the status code containing the last error (if any) from a  *)
(* previous operation.                                                *)

PROCEDURE InstallErrorHandler (handler: ErrorHandler);
(* Installs client-module error procedure to handle errors that occur *)
(* during operations.  Until an error-handler is installed, all       *)
(* errors will be ignored.                                            *)

PROCEDURE SizeOfStatic (string: ARRAY OF CHAR): CARDINAL;
(* Return size of static string plus terminating null character.      *)
```

```
PROCEDURE SizeOfDyna (dynaStr: DynamicString): CARDINAL;
(* Return size of dynamic string not including terminating null char. *)

PROCEDURE Assign (string: ARRAY OF CHAR; VAR dynaStr: DynamicString);
(* Assign static string to dynamic string.                            *)

PROCEDURE Dispose (VAR dynaStr : DynamicString);
(* Dispose of a dynamic string.                                       *)

PROCEDURE Change (string: ARRAY OF CHAR; VAR dynaStr: DynamicString);
(* Changes (updates) dynamic string.                                  *)

PROCEDURE Append (VAR r: DynamicString; s: DynamicString);
(* Append dynamic string s to dynamic string r.                       *)

PROCEDURE Concat (VAR r: DynamicString; s1, s2: DynamicString);
(* Concatenates s1 and s2 with the result in r.                       *)

PROCEDURE Copy (VAR r: DynamicString; s: DynamicString);
(* Copies dynamic string s to dynamic string r.                       *)

PROCEDURE Delete (VAR dynaStr: DynamicString; p, l: CARDINAL);
(* Deletes a sequence of l characters in dynamic string starting at   *)
(* position p.  Delete has no effect if p is greather than the size   *)
(* of the dynamic string.                                             *)

PROCEDURE Insert (VAR r: DynamicString; s: DynamicString; p: CARDINAL);
(* Inserts dynamic string s into dynamic string r at position p.  If  *)
(* p is greater than the size of the dynamic string, then s will be   *)
(* appended to r.                                                     *)

PROCEDURE Pos (s, p: DynamicString): CARDINAL;
(* Returns the position (starting from 0) of the first occurance of   *)
(* the dynamic substring p in the dynamic string s.  If p is not in s *)
(* then the result is MAX (CARDINAL).                                 *)

PROCEDURE Slice (VAR r: DynamicString; s: DynamicString; p, l: CARDINAL);
(* r is assigned a slice of dynamic string s.  This slice goes from   *)
(* position p to p + l.  If p + l is greater than the size of s, then *)
(* r is assigned the slice from p to size of s.  If p is greater than *)
(* the size of s or l is 0 then r becomes null.                       *)

PROCEDURE MakeStatic (dynaStr: DynamicString; VAR string: ARRAY OF CHAR);
(* Transfer dynamic string to string buffer.  String buffer should be *)
(* large enough to hold dynamic string; if string is not large enough *)
(* then only a partial transfer is made.                              *)

PROCEDURE Compare (string: ARRAY OF CHAR; dynaStr: DynamicString): INTEGER;
(* Compare static string to dynamic string.                           *)
(* Returns 0 if s1 = s2; -1 if s1 < s2; +1 if s1 > s2.                *)

PROCEDURE Equal (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string equals dynamic string.               *)

PROCEDURE Greater (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string is greater than dynamic string.      *)
```

```
PROCEDURE Less (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string is less than dynamic string.          *)

PROCEDURE CompareDyna (d1, d2: DynamicString): INTEGER;
(* Compare dynamic strings d1 and d2 lexicographically.                *)
(* Returns 0 if d1 = d2; -1 if d1 < d2; +1 if d1 > d2.                 *)

PROCEDURE EqualDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 = d2.                                           *)

PROCEDURE GreaterDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 > d2.                                           *)

PROCEDURE LessDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 < d2.                                           *)

PROCEDURE TrimLeft (VAR dynaStr: DynamicString; trim: CHAR);
(* Delete leading trim characters from dynamic string.                *)

PROCEDURE TrimRight (VAR dynaStr: DynamicString; trim: CHAR);
(* Delete trailing trim characters from dynamic string.               *)

PROCEDURE TrimAll (VAR dynaStr: DynamicString; trim: CHAR);
(* Trims all leading and trailing trim characters from dynamic string *)

PROCEDURE WrDynaStr (dynaStr: DynamicString);
(* Writes dynaChar to STDOUT.                                         *)

PROCEDURE RdDynaStr (VAR dynaStr: DynamicString);
(* Read dynamic string from STDIN.  Maximum bytes that can be read at *)
(* one time is 253 bytes.                                             *)

PROCEDURE FIOWrDynaStr (fh: File; dynaStr: DynamicString);
(* Writes dynamic string to file handle fh.                           *)

PROCEDURE FIORdDynaStr (fh: File; VAR dynaStr: DynamicString);
(* Reads in dynamic string from file handle fh.                       *)

END PMPDyna.
```

```
(*=============================================================
  =  Program Id:     PMPDYNA.MOD                              =
  =  Programmer:     Peter M. Perchansky                      =
  =  Copyright:      All rights reserved, Peter M. Perchansky, 1990  =
  =  Purpose:        Export dynamic string types and procedures.     =
  =  Date Written:   03-05-90                                 =
  =============================================================*)

IMPLEMENTATION MODULE PMPDyna;

(*-------------- Procedures from JPI's TopSpeed Modula II ------------*)
              IMPORT AsmLib;                    (* for AsmLib.Compare *)
FROM AsmLib  IMPORT DecAddr, Dos, Fill, IncAddr, Move, ScanR;
FROM AsmLib  IMPORT ScanNeL, ScanNeR;
FROM FIO     IMPORT File, RdStr, WrChar;
               IMPORT IO;                              (* for IO.WrChar      *)
FROM Storage IMPORT Available, ALLOCATE, DEALLOCATE;
FROM SYSTEM  IMPORT ADR, ADDRESS, Ofs, Registers, Seg;
(*-------------------------------------------------------------------*)


CONST
    maxCard = MAX (CARDINAL);
    null    = CHR (0);

TYPE
    DynamicString = ADDRESS;                    (* opaque type defined. *)


(*-------------------------------------------------------------------*)
(*          Global Variables used internally by PMPDyna.             *)
(*-------------------------------------------------------------------*)

VAR
    CallError : ErrorHandler; (* client-module installed error proc   *)
    lastError : DynaStat;     (* set by procedures when error occurs  *)
    noError   : BOOLEAN;      (* set true if when errors are ignored  *)


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

PROCEDURE Assign (string: ARRAY OF CHAR; VAR dynaStr: DynamicString);
FORWARD;
(* FORWARD reference to Assign.  Enables SetError to call Assign.     *)

PROCEDURE SetError (status: DynaStat; loc: ARRAY OF CHAR);
(* Sets lastError to status and calls client-installed error handler. *)

VAR
    dynaLoc : DynaLoc;

BEGIN
    IF NOT noError THEN
       lastError := status;              (* udate last error status. *)
       Assign (loc, dynaLoc);            (* create dynamic string.   *)
       CallError (status, dynaLoc);      (* call the error handling  *)
    END;                                 (* procedure.               *)
END SetError;

PROCEDURE IgnoreError (status: DynaStat; loc: DynaLoc);
(* InstallErrorHandler initially sets CallError to this procedure.    *)
(* Until the Client-module installs its own error handling procedure, *)
(* all errors will be ignored.                                        *)

BEGIN
END IgnoreError;
```

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

PROCEDURE InitDyna (VAR dynaStr: DynamicString);
(* Initialize dynamic string for use.  Should be called prior to any  *)
(* operation on a "new" (ie. never assigned) dynamic string varaiable *)
(* This procedure should not be used on dynamic strings that were     *)
(* assigned (use Dispose instead).                                    *)

BEGIN
    dynaStr := NIL;
END InitDyna;

PROCEDURE LastError (): DynaStat;
(* Returns the status code containing the last error (if any) from a  *)
(* previous operation.                                                *)

BEGIN
    RETURN lastError;
END LastError;

PROCEDURE InstallErrorHandler (handler: ErrorHandler);
(* Installs client-module error procedure to handle errors that occur *)
(* during operations.  Until an error-handler is installed, all       *)
(* errors will be ignored.                                            *)

BEGIN
    CallError := handler;
END InstallErrorHandler;

PROCEDURE SizeOfStatic (string: ARRAY OF CHAR): CARDINAL;
(* Return size of static string plus terminating null character.      *)

BEGIN
    RETURN (ScanR (ADR (string), HIGH (string) + 1, 0) + 1);
END SizeOfStatic;

PROCEDURE SizeOfDyna (dynaStr: DynamicString): CARDINAL;
(* Return size of dynamic string not including terminating null char. *)

VAR
    size : CARDINAL;

BEGIN
    size := 0;
    IF dynaStr # NIL THEN
       size := ScanR (dynaStr, maxCard, 0)
    END;
    RETURN size;
END SizeOfDyna;

PROCEDURE Assign (string: ARRAY OF CHAR; VAR dynaStr: DynamicString);
(* Assign static string to dynamic string.                            *)

VAR
    dataSize  : CARDINAL;

BEGIN
    string[HIGH (string)] := null;
    dataSize := SizeOfStatic (string);
    IF Available (dataSize) THEN
       ALLOCATE (dynaStr, dataSize);
       Move (ADR (string), dynaStr, dataSize);
    ELSE
       SetError (full, "Assign ")
    END;
END Assign;

PROCEDURE Dispose (VAR dynaStr : DynamicString);
(* Dispose of a dynamic string.                                       *)

VAR
    dataSize : CARDINAL;

BEGIN
    dataSize := SizeOfDyna (dynaStr);
    IF (dataSize > 0) AND (dynaStr # NIL) THEN
       DEALLOCATE (dynaStr, dataSize + 1);    (* include null in size *)
       dynaStr := NIL;
    ELSE
       SetError (empty, "Dispose ")
    END;
END Dispose;

PROCEDURE Change (string: ARRAY OF CHAR; VAR dynaStr: DynamicString);
(* Changes (updates) dynamic string.                                  *)

    PROCEDURE Update (VAR d: DynamicString; string: ARRAY OF CHAR);

    BEGIN
        noError := TRUE;
        Dispose (d);
        Assign (string, d);
        noError := FALSE;
    END Update;

VAR
    stringSize,
    dynaSize    : CARDINAL;

BEGIN
    stringSize := SizeOfStatic (string);
    dynaSize := SizeOfDyna (dynaStr) + 1; (* include terminating null *)
    IF stringSize > dynaSize THEN
       IF Available (stringSize - dynaSize) THEN
          Update (dynaStr, string)
       ELSE
          SetError (full, "Change ")
       END;
    ELSE
       Update (dynaStr, string);
    END;
END Change;

PROCEDURE Append (VAR r: DynamicString; s: DynamicString);
```

```
(* Append dynamic string s to dynamic string r.                    *)

VAR
    dataSize,
    rSize,
    sSize    : CARDINAL;
    temp     : DynamicString;

BEGIN
    rSize := SizeOfDyna (r);        (* does not include null char     *)
    sSize := SizeOfDyna (s) + 1;    (* include terminating null char  *)
    dataSize := rSize + sSize;
    IF Available (dataSize) THEN
       ALLOCATE (temp, dataSize);   (* create temporary work area.    *)
       Move (r, temp, rSize);       (* move receiving string - null.  *)
       IncAddr (temp, rSize);       (* move address past receiving st *)
       Move (s, temp, sSize);       (* append second string incl null *)
       DecAddr (temp, rSize);       (* go back to beginning of block  *)
       noError := TRUE;
       Dispose (r);                 (* release memory of receiver.    *)
       noError := FALSE;
       ALLOCATE (r, dataSize);      (* create new receiver            *)
       Move (temp, r, dataSize);    (* move work buffer to receiver   *)
       DEALLOCATE (temp, dataSize); (* erase work buffer              *)
    ELSE
       SetError (full, "Append ")
    END;
END Append;

PROCEDURE Concat (VAR r: DynamicString; s1, s2: DynamicString);
(* Concatenates s1 and s2 with the result in r.                    *)

VAR
    dataSize    : CARDINAL;

BEGIN
    dataSize := SizeOfDyna (s1) + SizeOfDyna (s2) + 1;
    IF Available (dataSize) THEN
       noError := TRUE;
       Dispose (r);
       Append (r, s1);
       Append (r, s2);
       noError := FALSE;
    ELSE
       SetError (full, "Concat ");
    END;
END Concat;

PROCEDURE Copy (VAR r: DynamicString; s: DynamicString);
(* Copies dynamic string s to dynamic string r.                    *)

VAR
    dataSize    : CARDINAL;

BEGIN
    dataSize := SizeOfDyna (s) + 1;
    IF Available (dataSize) THEN
       noError := TRUE;
       Dispose (r);
       Append (r, s);
       noError := FALSE;
    ELSE
       SetError (full, "Copy ");
    END;
END Copy;

PROCEDURE Delete (VAR dynaStr: DynamicString; p, l: CARDINAL);
(* Deletes a sequence of l characters in dynamic string starting at   *)
(* position p.  Delete has no effect if p is greather than the size   *)
(* of the dynamic string.                                             *)

VAR
    temp      : DynamicString;
    dSize,
    dataSize,
    i         : CARDINAL;

BEGIN
    dSize := SizeOfDyna (dynaStr) + 1;  (* add terminating null char. *)
    IF p < dSize THEN
       IF l < dSize - p THEN
          i := p + l;
          IF Available (dSize) THEN
             Copy (temp, dynaStr);        (* copy to work buffer.     *)
             IncAddr (temp, p);           (* advance past p chars.    *)
             IncAddr (dynaStr, i);        (* advance past i chars.    *)

             REPEAT
                  Move (dynaStr, temp, 1);        (* move one char.   *)
                  INC (p); IncAddr (temp, 1);     (* increment index. *)
                  INC (i); IncAddr (dynaStr, 1);  (* increment index. *)
             UNTIL i = dSize;

             DecAddr (dynaStr, i);       (* reset addresses to start   *)
             DecAddr (temp, p);          (* of memory block.           *)
             Dispose (dynaStr);          (* get rid of receiver.       *)
             dataSize := dSize - l;      (* subtract chars deleted.    *)
             ALLOCATE (dynaStr, dataSize); (* create new receiver.     *)
             Move (temp, dynaStr, dataSize); (* move work to receiver *)
             Dispose (temp);             (* dispose of work buffer.    *)
          ELSE
             SetError (full, "Delete ")
          END;
       ELSE
          Dispose (dynaStr);             (* deleted all characters.    *)
       END;
    END;
END Delete;

PROCEDURE Insert (VAR r: DynamicString; s: DynamicString; p: CARDINAL);
(* Inserts dynamic string s into dynamic string r at position p.  If  *)
(* p is greater than the size of the dynamic string, then s will be   *)
(* appended to r.                                                     *)

VAR
    temp      : DynamicString;
```

```
    rSize,
    sSize,
    dataSize : CARDINAL;

BEGIN
    rSize := SizeOfDyna (r) + 1;    (* include terminating null char. *)
    sSize := SizeOfDyna (s);        (* not including terminating null *)
    IF p >= rSize THEN
       Append (r, s)
    ELSE
       dataSize := rSize + sSize;
       IF Available (dataSize) THEN
          ALLOCATE (temp, dataSize);   (* create temp. work buffer    *)
          Move (r, temp, p);           (* move p # chars into buffer  *)
          IncAddr (temp, p);           (* inc buffer past p chars.    *)
          Move (s, temp, sSize);       (* move s into temp.           *)
          IncAddr (temp, sSize);       (* inc buffer past s.          *)
          IncAddr (r, p);              (* inc receiver past p chars.  *)
          Move (r, temp, rSize - p);   (* move rest of receiver.      *)
          DecAddr (r, p);              (* reset r to start of block.  *)
          DecAddr (temp, p + sSize);   (* reset temp to start.        *)
          Dispose (r);                 (* release memory from rec.    *)
          ALLOCATE (r, dataSize);      (* create new receiver.        *)
          Move (temp, r, dataSize);    (* move buffer to receiver.    *)
          DEALLOCATE (temp, dataSize); (* release work buffer.        *)
       ELSE
          SetError (full, "Insert ")
       END;
    END;
END Insert;

PROCEDURE Pos (s, p: DynamicString): CARDINAL;
(* Returns the position (starting from 0) of the first occurance of   *)
(* the dynamic substring p in the dynamic string s.  If p is not in s *)
(* then the result is MAX (CARDINAL).                                 *)

VAR
    sSize,
    pSize,
    i, j, k : CARDINAL;

BEGIN
    sSize := SizeOfDyna (s) + 1;     (* include terminating null char *)
    pSize := SizeOfDyna (p) + 1;     (* include terminating null char *)
    i := 0;
    LOOP
       IF (i > sSize) OR (CHAR (s^) = null) THEN
          RETURN maxCard
       END;
       j := 0;
       k := i;
       LOOP
          IF (j > pSize) OR (CHAR (p^) = null) THEN
             RETURN i
          END;
          IF k > sSize THEN
             RETURN maxCard
          END;
          IF CHAR (s^) # CHAR (p^) THEN
             EXIT
          END;

          INC (j); IncAddr (p, 1);
          INC (k); IncAddr (s, 1);
       END;
       INC (i); IncAddr (s, 1);
    END;
END Pos;

PROCEDURE Slice (VAR r: DynamicString; s: DynamicString; p, l: CARDINAL);
(* r is assigned a slice of dynamic string s.  This slice goes from   *)
(* position p to p + l.  If p + l is greater than the size of s, then *)
(* r is assigned the slice from p to size of s.  If p is greater than *)
(* the size of s or l is 0 then r becomes null.                       *)

VAR
    sSize,
    dataSize : CARDINAL;

BEGIN
    noError := TRUE;
    Dispose (r);
    noError := FALSE;
    sSize := SizeOfDyna (s) + 1; (* include terminating null char.    *)
    IF Available (sSize - p + 1) THEN
       IF NOT (p > sSize) THEN
          IF NOT (p + l > sSize) THEN
             dataSize := l + 1;   (* include 1 for the null character *)
             ALLOCATE (r, dataSize);
             Fill (r, dataSize, 0);     (* make life easy - null fill *)
             IncAddr (s, p);            (* move past p characters     *)
             Move (s, r, l);            (* slice off chars into s     *)
          ELSE
             dataSize := sSize - p + 1; (* starting from p to end     *)
             ALLOCATE (r, dataSize);
             Fill (r, dataSize, 0);     (* null fill.                 *)
             IncAddr (s, p);            (* move past p characters     *)
             Move (s, r, dataSize - 1);
          END;
       END;
    ELSE
       SetError (full, "Slice ")
    END;
END Slice;

PROCEDURE MakeStatic (dynaStr: DynamicString; VAR string: ARRAY OF CHAR);
(* Transfer dynamic string to string buffer.  String buffer should be *)
(* large enough to hold dynamic string; if string is not large enough *)
(* then only a partial transfer is made.                              *)

VAR
    dataSize : CARDINAL;

BEGIN
    Fill (ADR (string), HIGH (string) + 1, 0);
    dataSize := SizeOfDyna (dynaStr) + 1;  (* include null character. *)
```

```
    IF dataSize > (HIGH (string) + 1) THEN
       dataSize := HIGH (string) + 1;
       SetError (mismatchedSize, "MakeStatic ");
    END;

    Move (dynaStr, ADR (string), dataSize);
END MakeStatic;

PROCEDURE Compare (string: ARRAY OF CHAR; dynaStr: DynamicString): INTEGER;
(* Compare static string to dynamic string.                          *)
(* Returns 0 if s1 = s2; -1 if s1 < s2; +1 if s1 > s2.               *)

VAR
    i : CARDINAL;

BEGIN
    i := 0;
    WHILE (CAP (string[i]) = CAP (CHAR (dynaStr^)))
      AND (string[i] # null) AND (CHAR (dynaStr^) # null) DO
          INC (i); IncAddr (dynaStr, 1);
    END;

    IF string[i] = CHAR (dynaStr^) THEN
       RETURN 0
    ELSIF string[i] < CHAR (dynaStr^) THEN
          RETURN -1
       ELSE
          RETURN +1
       END;
END Compare;

PROCEDURE Equal (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string equals dynamic string.              *)

BEGIN
    RETURN (Compare (string, dynaStr) = 0);
END Equal;

PROCEDURE Greater (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string is greater than dynamic string.     *)

BEGIN
    RETURN (Compare (string, dynaStr) = 1);
END Greater;

PROCEDURE Less (string: ARRAY OF CHAR; dynaStr: DynamicString): BOOLEAN;
(* Returns true if static string is less than dynamic string.        *)

BEGIN
    RETURN (Compare (string, dynaStr) = -1);
END Less;

PROCEDURE CompareDyna (d1, d2: DynamicString): INTEGER;
(* Compare dynamic strings d1 and d2 lexicographically.              *)
(* Returns 0 if d1 = d2; -1 if d1 < d2; +1 if d1 > d2.               *)

VAR
    d1Size,
    d2Size,
    size, i : CARDINAL;

BEGIN
    d1Size := SizeOfDyna (d1);
    d2Size := SizeOfDyna (d2);
    IF d1Size < d2Size THEN
       size := d1Size
    ELSE
       size := d2Size
    END;

    i := AsmLib.Compare (d1, d2, size);
    IF i = size  THEN
       RETURN 0
    ELSIF i < size THEN
          IncAddr (d1, i);
          IncAddr (d2, i);

          IF CHAR (d1^) < CHAR (d2^) THEN
             RETURN -1
          ELSE
             RETURN +1
          END;

       ELSIF (d1Size = d2Size) THEN
             RETURN 0
          ELSIF (d1Size < d2Size) THEN
                RETURN -1
             ELSE
                RETURN 1
             END;
END CompareDyna;

PROCEDURE EqualDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 = d2.                                          *)

BEGIN
    RETURN (CompareDyna (d1, d2) = 0);
END EqualDyna;

PROCEDURE GreaterDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 > d2.                                          *)

BEGIN
    RETURN (CompareDyna (d1, d2) = +1);
END GreaterDyna;

PROCEDURE LessDyna (d1, d2: DynamicString): BOOLEAN;
(* Returns true if d1 < d2.                                          *)

BEGIN
    RETURN (CompareDyna (d1, d2) = -1);
END LessDyna;

PROCEDURE TrimLeft (VAR dynaStr: DynamicString; trim: CHAR);
(* Delete leading trim characters from dynamic string.               *)
```

```
VAR
    count, dataSize : CARDINAL;

BEGIN
    dataSize := SizeOfDyna (dynaStr);
    count := ScanNeR (dynaStr, dataSize, SHORTCARD (ORD (trim)));
    IF count > 0 THEN
       Delete (dynaStr, 0, count)
    END;
END TrimLeft;

PROCEDURE TrimRight (VAR dynaStr: DynamicString; trim: CHAR);
(* Delete trailing trim characters from dynamic string.              *)

VAR
    count, dataSize : CARDINAL;

BEGIN
    dataSize := SizeOfDyna (dynaStr);
    IncAddr (dynaStr, dataSize - 1);    (* setup dynaStr for ScanNeL *)
    count := ScanNeL (dynaStr, dataSize, SHORTCARD (ORD (trim)));
    DecAddr (dynaStr, dataSize - 1);    (* restore dynaStr to start  *)
    Delete (dynaStr, dataSize - count, count);
END TrimRight;

PROCEDURE TrimAll (VAR dynaStr: DynamicString; trim: CHAR);
(* Trims all leading and trailing trim characters from dynamic string *)

BEGIN
    TrimLeft (dynaStr, trim);
    TrimRight (dynaStr, trim);
END TrimAll;

PROCEDURE WrDynaStr (dynaStr: DynamicString);
(* Writes dynaStr to STDIN.                                          *)

BEGIN
    WHILE (dynaStr # NIL) AND (CHAR (dynaStr^) # null) DO
          IO.WrChar (CHAR (dynaStr^));
          IncAddr (dynaStr, 1);
    END;
END WrDynaStr;

PROCEDURE RdDynaStr (VAR dynaStr: DynamicString);
(* Read dynamic string from STDIN.  Maximum bytes that can be read at *)
(* one time is 253 bytes.                                            *)

CONST
    maxBuff = 253;

VAR
    r      : Registers;
    buffer : RECORD
               bytesToRead : CHAR;
               bytesRead   : CHAR;
               input       : ARRAY [0..maxBuff] OF CHAR;
             END;
```

```
BEGIN
    Fill (ADR (buffer.input), maxBuff + 1, 0);
    buffer.bytesToRead := CHR (maxBuff);
    buffer.bytesRead := null;
    WITH r DO
         AH := 0AH;                                  (* buffered input *)
         DS := Seg (buffer);
         DX := Ofs (buffer);
    END;
    Dos (r);
    Assign (buffer.input, dynaStr);
END RdDynaStr;

PROCEDURE FIOWrDynaStr (fh: File; dynaStr: DynamicString);
(* Writes dynamic string to file handle fh.                          *)

BEGIN
    WHILE (dynaStr # NIL) AND (CHAR (dynaStr^) # null) DO
          WrChar (fh, CHAR (dynaStr^));
          IncAddr (dynaStr, 1);
    END;
END FIOWrDynaStr;

PROCEDURE FIORdDynaStr (fh: File; VAR dynaStr: DynamicString);
(* Reads in dynamic string from file handle fh.                      *)

VAR
    buffer : POINTER TO ARRAY [0..maxCard] OF CHAR;

BEGIN
    NEW (buffer);
    RdStr (fh, buffer^);
    Assign (buffer^, dynaStr);
    DISPOSE (buffer);
END FIORdDynaStr;

BEGIN                                          (* Module initialization *)
    InstallErrorHandler (IgnoreError);
    lastError := none;
    noError := FALSE;
END PMPDyna.
```

Two Ways To Shuffle

by Tom Cattrall

Occasionally the need arises to shuffle a list of items. It is a simple task but often the first algorithm that people come up with for the task isn't especially simple or fast. Below are 2 different methods for shuffling that I've come across.

The following discussion uses a deck of cards as an example but the technique can be used for any similar application where you need to shuffle a list of items. The deck is represented as:

```
aCard = RECORD
          suit : aSuit;
          value : aCardValue;
        END;
aDeck = ARRAY [ 1..52 ] OF aCard;

PROCEDURE InitDeck( VAR deck : aDeck);
VAR
    s : aSuit;
    v : aCardValue;
BEGIN
  FOR s := clubs TO spades DO
    FOR v := 1 TO 13 DO
      WITH deck[ 13 * ORD(s) + v ] DO
        BEGIN
          suit := s;
          value := v;
        END;
 END;
```

To shuffle the deck implies that you rearrange them in place. To deal the deck means that you randomly pick elements out of the deck until they are all gone. If the deck is first shuffled, you can deal the deck by using a for loop to step through the 52 entries in sequence.

Method 1:

This method of shuffling calls a random number routine to return a number between 1 and 51. The card chosen by that number is swapped with card number 52. The next card is chosen using a random number call to return a value between 1 and 50 and it is exchanged with the card in location 51. Repeat this until all 52 have been processed. The shuffle code for this method looks like:

```
PROCEDURE ShuffleDeck( VAR deck : aDeck);
VAR
    i,
    j  : INTEGER;
    t  : aCard;
BEGIN
  FOR i := 52 DOWNTO 2 DO
    BEGIN
      t := deck [i];
      j := RandomInt(i - 1);
      deck[i] := deck[j];
      deck[j] := t;
    END;
END;
```

Method 2:

Another method is to add an auxiliary number to the definition of aCard. This number is initialized using calls to a random number generator. Then you sort the array using this value as the key. At the end of the sort the auxiliary numbers are in order but the cards themselves are now in random order.

The record is changed as shown here:

```
aCard = RECORD
          suit : aSuit;
          value : aCardValue;
          auxValue : INTEGER;
        END;
```

A line is added in the WITH statement of InitDeck that assigns auxValue := RandomInt(52). ShuffleDeck can be any sort that is appropriate to the size of the deck being used. For 52 elements, the following insertion sort is adequate. If the size is much larger then a more powerful sort such as Shell sort or quick sort would be a good idea.

```
PROCEDURE ShuffleDeck( VAR deck : aDeck );
VAR
    t  : aCard;
    i,
    j  : INTEGER;

BEGIN
  FOR i := 2 TO 52 DO
    BEGIN
      j := i;
      IF deck[j].auxValue < deck[1].auxValue THEN
        BEGIN
          t := deck[1];
          deck[1] := deck[j];
          deck[j] := t;
        END;
      t := deck[j];
      WHILE (deck[j-1].auxValue > t.auxValue) DO
        BEGIN
          deck[j] := deck[j-1];
          j := j - 1;
        END;
      deck[j] := t;
    END;
END;
```

The first algorithm is quite a bit faster. On my machine, it took 1.2 milliseconds to shuffle 52 entries, while the second algorithm took 3.1msecs. These times are only for the ShuffleDeck routines. The second algorithm would probably be faster if a better sort such as QuickSort were used, but it would be even more complicated. So, it looks like the first algorithm scores best on both simplicity and speed.

A Review of Apple's Lisa WorkShop

David Craig
736 Edgewater, Wichita KS 67230

(October 12, 1988)

INTRODUCTION

The WorkShop was Apple's software development environment for the Lisa during the years 1981 to 1985. From 1981 to 1983 this environment produced all the code for the Lisa. After Apple's Macintosh introduction in 1984 Macintosh developers used the WorkShop since no native Macintosh development environment existed.

The WorkShop is a command line shell similar in appearance to Apple's older Apple // and /// UCSD–like Pascal shells. The main command line provides access to two other command lines, the File Manager and the System Manager, and to several programming tools. The programming tools consist of an editor, Pascal compiler, 68000 assembler, and a linker. Even though the WorkShop supports various languages it is really tailored toward Pascal since most of its utilities are for Pascal source code and object files. The main command line appears as

```
{V3.9} WORKSHOP: FILE-MGR, SYSTEM-MGR, Edit, Run, Debug, Pascal, Basic, Quit, ?
```

with the other half of this line being

```
Assemble, Generate, MakeBackground, Link, TransferProgram
```

FILE MANAGER

The File Manager provides access to files which are stored either on 400K microdiskettes, hard disks, or tape backup systems. The Lisa Operating System supports a hierarchical file structure which the WorkShop also supports. The command line for the File Manager appears as

```
FILE-MGR:  Backup, Copy, Delete, List, Online, Prefix, Rename, Transfer, Quit, ?
```

with the other half of this line being

```
AddCatalog, Equal, FileAttributes, Initialize, Mount, Names, Scavenge, Unmount
```

With this command line you could backup, copy, and delete files. Wildcard characters within file names are supported for easier file selection. For a directory listing the **List** command displays a listing complete with file sizes, creation and modification dates, and file attributes. A sample follows:

```
Filename                  Size Psize   Last-Mod-Date    Creation-Date   Attr
--------                  ---- -----   -------------    -------------   ----
CARTOG.OBJ                2560     5   10/02/88-10:53   10/02/88-10:52  SC
CARTOG.TEXT              15360    30   09/29/88-18:20   12/03/87-23:40  O
CARTOG/backup.TEXT        2048     4   10/01/88-11:22   09/28/88-22:01
CARTOG/Map_Util.OBJ      34304    67   10/09/88-15:26   10/09/88-15:25
CARTOG/Map_Util.TEXT     50176    98   10/08/88-16:33   09/11/88-17:05  P
CARTOG/Map_UtilX.TEXT     4096     8   09/29/88-22:17   09/11/88-17:37
CARTOG/UNewMapMgr.OBJ    47104    92   10/09/88-15:22   10/09/88-15:22
CARTOG/UNewMapMgr.TEXT   82944   162   10/09/88-15:20   09/28/88-09:48
CARTOG/UOldMapMgr.OBJ    15872    31   10/09/88-15:21   10/09/88-15:21
CARTOG/UOldMapMgr.TEXT   32768    64   10/08/88-16:34   09/11/88-18:21
CARTOGX.TEXT              2048     4   09/29/88-14:41   09/08/88-23:37
```

```
565 total blocks for files listed
94 blocks of OS overhead for volume and files listed
1115 blocks free out of 17418
```

The attributes field specifies the attributes of a file. These are defined as follows:

```
O - File is currently open
C - File was closed by the Operating System
L - File has its Safety flag set (i.e., is locked)
P - File is Protected from copying
S - File was scavenged by the disk Scavenger
```

The **Online** command displays a list of the currently mounted devices as follows:

DevName	DevAlias	VolumeName	VolSize	FreeBlks	Files	Open	Attr
#12	UPPER	AOS 3.0	17418	1113	244	42	MBP
#13	LOWER	Lisa Sony Disk	772	762	4	0	M
#15#1	ALTCONSOLE		0	0	0	0	M
#15#2	MAINCONSOLE		0	0	0	1	M
#10#1	RS232A	<printer>	0	0	0	0	M
#2#1	SLOT2CHAN1	ProFile	9690	2714	189	0	M

The **AddCatalog** command creates a new subdirectory, **Equal** compares files for equality, and **Initialize** formats disk volumes. The **Scavenge** command analyses disk volumes for irregularities and repairs them if any problems exist. **Mount** and **Unmount** make external devices visible to the Lisa Operating System and the WorkShop. The **FileAttributes** command produces the following command line:

```
File Attributes: ClearAttributes, Protect, Safety, AddPassword, RemovePassword, Quit
```

This command line deletes or specifies new file attributes. The **Safety** command marks a file as non–deletable so that if you try and delete it the delete will fail. This attribute is very useful for important files. **Protect** marks a file as protected so that the file can never be copied. This attribute was used by the Lisa Office System (a.k.a. Lisa 7/7) to turn programs into "protected masters". **AddPassword** and **RemovePassword** allow file password protection.

SYSTEM MANAGER

The System Manager provides access to several low–level features of the Lisa. Its command line is

```
SYSTEM-MGR:  ManageProcess, OutputRedirect, Preferences, Time, Quit, ?
```

with the second half appearing as

```
Console, FilesPrivate, Validate, DefaultPrinter
```

ManageProcess lists all the current system processes and can terminate executing processes. **OutputRedirect** redirects all console output to a text file whose name you specify. **Preferences** runs the Lisa Preferences tool which appears as a window with buttons that are mouse controlled. This window allows you to specify several hardware parameters of the Lisa such as the screen brightness, speaker volume level, and keyboard and mouse sensitivity. **Time** displays the current clock date and time. **Console** allows the WorkShop's main console I/O to originate from either the alternate console or an external terminal connected to one of the Lisa's serial ports. **FilesPrivate** enables the File Manager to have access to special Office System files whose names start with "{".

Generally WorkShop programs should not access these files since their integrity is vital for correct operation of the Lisa Office System.

The **Validate** command controls several of what are called "paranoia" settings, such as whether file transfers will be verified and whether file selections will require user confirmation for File Manager operations. This allows you to tailor the system to your level of confidence in it and in yourself.

PROGRAMMING TOOLS

The Lisa WorkShop supports several powerful programming tools. Access to these is available through the main command line. The commands in the line for the tools are

```
Edit, Debug, Pascal, Basic, Assemble, Generate, Link
```

The first tool that is usually used is the source code editor. The editor, called **LisaEdit**, provides full window, menu bar, and mouse support. Menus exist in a menu bar which supports pull–down menus. These handle file creation, opening, saving, and printing. In an early version of LisaEdit the Print menu supported underlining of Pascal keywords. Unfortunately, later versions eliminated this handy feature. Multiple overlapping and resizable windows are available for editing source code files with file size limited only by the amount of memory your Lisa has. With my 1 Mbyte system I have had been able to work with about a dozen large programs. Text resizing is available through the Type Style menu. You can have very small text with about 150 characters per line (cpl) to very large text with about 60 cpl. Cutting and pasting is done with the mouse and the Edit menu which supports the commands Cut, Copy, and Paste. The Lisa's arrow keys on the keypad are supported for cursor movement. The Undo menu command provides the ability to "undo" your last operation. For example, if you Cut some text that was not supposed to be cut Undo will undo the Cut. When you leave LisaEdit and return to the WorkShop main command line the opened windows in LisaEdit remain active. Later, when you reenter LisaEdit the windows appear automatically. This is very handy for modifying a program, leaving LisaEdit to compile the program, and returning to LisaEdit.

The **Pascal Compiler** is the heart of the Lisa WorkShop. For a detailed review of this compiler's features see my paper titled "A Review of Apple's Lisa Pascal". This compiler generates I–code (intermediate code) which is actually only standard UCSD Pascal P–code. If an error occurs during a compilation the editor is run, the source file is loaded, the cursor is placed over the offending statement, and a specific error message is displayed. The **Code Generator**, which is run automatically by the Compiler, takes an I–code file and produces 68000 object code. The **Linker** links different files and libraries to create executable object files. The Compiler and Code Generator display a lot of compilation statistics as the following example shows:

```
Lisa Pascal Compiler V3.76 (05-Apr-85)                  10:50:46 10-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.

Input file - [.TEXT] RangeTester
List file - [.TEXT]
Output file - [RangeTester] [.OBJ]

[242109 words] RANGE_TE

Elapsed time: 5.901 seconds.
Compilation complete - no errors found.  9 lines.

Lisa Pascal MC68000 Code Generator V3.65 (20-Mar-85) 10:51:00 10-Oct-88
(c)1981 SVS, Inc.  (c)1983, 1984 Apple Computer, Inc.
```

```
Input file - $I+
Input file - [.I] RangeTester
Output file - [RangeTester] [.OBJ] RangeTester

RANGE_TE - RANGE_TE    Code size =      88

Elapsed time: 2.580 seconds.
Total code size = 088

Linker - M68000 Object Code   v0.9.3.1 08-Apr-85 15:26:15
Copyright Apple Computer, Inc. 1985

Beginning memory:           269656
After initial allocation:   233146
Input file [.OBJ] ? ?
Options ? ?
Options are:

      Option Value Description:
      ------ ----- ------------
      +A -A    '-'  Alphabetical Listing
      +C MODNAME SegName    Copy module into segment
      +F -F    '-'  For domain 0
      -H num    Initial Stack Swap Area:    4096
      +I -I    '+'  Interfaces Flag
      +L -L    '-'  Location Ordered Listing
      +M fromName toName       Segment Name Mapping
      +O -O    '+'  Emit O.S. Data record
      +P -P    '-'  Physical Link, machines w/out MMU's.
      +R -R    '-'  MODNAME Do partial link
                      If MODNAME is provided,
                      dead code will be stripped using it as the root
      +S num        Start Dynamic Stack Size:   10240
      +T num        Top Dynamic Stack Size:    131072
      +W Which directory file: -#12-INTRINSIC.LIB
      +X -X    '-'  Cross-develop to MAC

Options ?
Input file [.OBJ] ? RangeTester
Input file [.OBJ] ? IOSPASLIB
Input file [.OBJ] ?
Listing file [-CONSOLE] / [.TEXT]
Output file ? [.OBJ] RangeTester
Reading file: RangeTester.OBJ
Reading file: IOSPASLIB.OBJ

Input summary:

    2 Files     , max =   100
    6 Segments  , max =  4096
   19 Modules   , max = 32768
   11 Entries   , max = 65536
    4 Ref. Lists, max = 65536
    4 References, max = 65536
```

```
Linking Main Program.
Reading Library Directory: -#12-INTRINSIC.LIB
Active : 1 of 19 read.
Visible: 1 of 11 read.
Global data: $000016
Common data: $000000
Number of segments in file = 1, number of Jump Table entries = 1
Linking segment:          file (JT) seg:    1 size:       88
0 Errors detected.

RangeTester.OBJ is an executable program file.

Elapsed time: 24.210 seconds.
That's all Folks!
```

The Pascal Compiler supports many directives which modify its behavior. For example, you can control integer and sub–range checking, short–circuit boolean evaluation, code optimization control (i.e., off, old scheme, new scheme), program routine name inclusion into the object code for debugging with LisaBug, assembly language listing, and conditional compilation. The last item is implemented using the following directives:

```
{$DECL var_name}  declares var_name to be a conditional variable
{$SETC var_name := expression} sets var_name to a boolean or an integer
{$IFC expression} tests if expression is true, if true includes following code
{$ELSEC} the else  part in the if-then-else-endif construct
{$ENDC}  the endif part in the if-then-else-endif construct
```

The assembly listing shows the generated assembly code listed after each source line. The listing for a simple Pascal program follows:

```
Lisa Pascal Compiler V3.76 (05-Apr-85)                    10:51:45 10-Oct-88
Lisa Pascal MC68000 Code Generator V3.65 (20-Mar-85)      10:51:57 10-Oct-88

1     1 --          PROGRAM Range_Tester; {$ASM+}
2     2 --
3     3 --            VAR a : ARRAY [0..9] OF INTEGER;
4     4 --                i : 0..10;
5     5 --
6     6 0-            BEGIN
          000000 4EBA 0000      RANGE_TE JSR     %_BEGIN
          000004 4E56 0000               LINK    A6,#$0000
          000008 2C5F                    MOVE.L  (A7)+,A6
          00000A 4E55 FFEA               LINK    A5,#$FFEA
          00000E 9FED 0010               SUBA.L  $0010(A5),A7
          000012 4EBA 0000               JSR     %_INIT
7     7 --              FOR i := 0 TO 10 DO
          000016 422D FFEB               CLR.B   $FFEB(A5)
          00001A 601A                    BRA.S   L0001           ; 00000036
8     8 --                a[i] := i;
          00001C 102D FFEB      L0002    MOVE.B  $FFEB(A5),D0
          000020 4880                    EXT.W   D0
          000022 41BC 0009               CHK     #$0009,D0
          000026 E340                    ASL.W   #$1,D0
          000028 122D FFEB               MOVE.B  $FFEB(A5),D1
          00002C 4881                    EXT.W   D1
```

```
                00002E 3B81 00EC                MOVE.W  D1,$EC(A5,D0.W)
                000032 522D FFEB                ADDQ.B  #$1,$FFEB(A5)
                000036 0C2D 000A FFEB L0001     CMPI.B  #$000A,$FFEB(A5)
                00003C 6FDE                     BLE.S   L0002           ; 0000001C
                00003E 4EBA 0000                JSR     %_TERM
                000042 4E5D                     UNLK    A5
                000044 4EBA 0000                JSR     %_END
                000048 4E75                     RTS
                00004A 4E5E                     UNLK    A6
                00004C 4E75                     RTS

                00004E D241 4E47 455F         .WORD   $D241,$4E47,$455F ; ".ANGE_"
                000054 5445                     .WORD   $5445             ; "TE"

                000056 0000           CstSize   .WORD   Last-CstSize-2
                000058                Last
9       9 -0            END.

Elapsed compilation time: 3.197 seconds.
Compilation complete - no errors found.  9 lines.
Elapsed code generator time: 4.075 seconds.
Total code size = 088
```

The **68000 Assembler** is used to create assembly routines which will later be linked to a Pascal program. Assembly is really used only for time or space critical code.

The Lisa WorkShop supports several other languages which must be purchased separately. A **C Compiler** was created shortly after the Lisa was introduced, but little programming effort seems to have been done with C due to the Lisa's overall Pascal perspective. **LisaBasic** was a rewrite of DEC's Basic–PLUS language, but since LisaBasic was an interpreter its use never caught on with programmers. **LisaCOBOL** appeared but was shortly forgotten. One interesting language which Apple developed for the Lisa but was never announced or released was **Magic/L** (pronounced "magical"). This language was a combination between Pascal and Forth, which I assume Apple Inc. found too exotic even for their revolutionary Lisa.

Debugging Lisa programs is accomplished in either of two ways. The preferred method for Pascal and other high level languages is to write debugging data to the Lisa's alternate console. This console, whose device name is "-ALTCONSOLE", is displayed when the Right–Option and Keypad–Enter keys are simultaneously pressed. While a program is running it displays debugging data on the alternate console using Pascal's WRITELN. The program's execution continues even when this console is displayed. To debug assembler or compiled code **LisaBug** exists. LisaBug uses the alternate console for both input and output. To debug a program use the Debug command from the main command line and you will be asked for the name of the file to debug. A breakpoint is set at the first instruction of the program and you can then single step, trace, or disassemble the program using various LisaBug commands. Routine names or symbols are available if the compiler's debugging directive, {$D+}, was enabled. When a runtime error occurs in a WorkShop program LisaBug is invoked. For example, the following Pascal program contains a runtime range error:

```
PROGRAM Range_Tester;

VAR a : ARRAY [0..9] OF INTEGER;
    i : 0..10;
```

```
BEGIN
  FOR i := 0 TO 10 DO    { <--- Runtime error occurs when i = 10 }
  a[i] := i;
END.
```

When this program runs the Lisa displays the following information on the alternate console and makes this console visible:

```
Level 7 Interrupt
RANGE_TE+0022 E340                    ASL.W #$1,D0
PC=0002002E SR=0000   0 US=00F7FAE6 SS=00CBFEE0 DO=1 P#=0008
D0=0000000A D1=00000009 D2=0000FFFF D3=00D00001
D4=00000004 D5=00CC9D54 D6=00F7D80C D7=00CC91E2
A0=A0220E1A A1=00CC91E2 A2=00CE004C A3=00CC9D42
A4=00CC9D42 A5=00F7FC4A A6=00F7FC4A A7=00F7FAE6

CHK RANGE ERROR in process of gid     8
sr =        0  pc =   131118
 saved registers at 13369278
Going to Lisabug, type g to continue
>
```

You are now in LisaBug and can issue LisaBug commands. For example, to see the program type "IL RANGE_TE" and the following appears:

```
>IL RANGE_TE
RANGE_TE+0000 4E56 0000      RANGE_TE LINK      A6,#$0000
RANGE_TE+0004 2C5F                    MOVE.L    (A7)+,A6
RANGE_TE+0006 4E55 FFEA               LINK      A5,#$FFEA
RANGE_TE+000A 9FED 0010               SUBA.L    $0010(A5),A7
RANGE_TE+000E A022 0238               IUJSR     $00220238          ; 00220238
RANGE_TE+0012 422D FFEB               CLR.B     $FFEB(A5)
RANGE_TE+0016 601A                    BRA.S     *+$001C            ; 0002003E
RANGE_TE+0018 102D FFEB               MOVE.B    $FFEB(A5),D0
RANGE_TE+001C 4880                    EXT.W     D0
RANGE_TE+001E 41BC 0009               CHK       #$0009,D0
RANGE_TE+0022 E340           PC       ASL.W     #$1,D0
RANGE_TE+0024 122D FFEB               MOVE.B    $FFEB(A5),D1
RANGE_TE+0028 4881                    EXT.W     D1
RANGE_TE+002A 3B81 00EC               MOVE.W    D1,$EC(A5,D0.W)
RANGE_TE+002E 522D FFEB               ADDQ.B    #$1,FFEB(A5)
RANGE_TE+0032 0C2D 000A FFEB          CMPI.B    #$000A,$FFEB(A5)
RANGE_TE+0038 6FDE                    BLE.S     *-$0020            ; 00020024
RANGE_TE+003A A022 0290               IUJSR     $00220290          ; 00220290
RANGE_TE+003E 4E5D                    UNLK      A5
RANGE_TE+0040 A022 0220               IUJSR     $00220220          ; 00220220
RANGE_TE+0044 4E75                    RTS
RANGE_TE+0046 4E5E                    UNLK      A6
RANGE_TE+0048 4E75                    RTS
```

This assembly listing shows in line RANGE_TE+0022 that the program crashed with a range error (CHK RANGE ERROR) since variable **i** was greater than 9, the last index in array **a**.

LisaBug can also be invoked at any time by pressing the "-" key on the keypad. This is useful when a program is in an infinite loop situation and nothing will make it stop. Once in LisaBug you can terminate the errant program and return to the WorkShop main command line.

For Macintosh programmers Apple developed the WorkShop Macintosh Software Supplement. This consisted of a set of Lisa diskettes containing Macintosh ToolBox and OS interfaces which allowed total access to every Macintosh feature. Various Macintosh oriented tools for the WorkShop were also provided. This included a resource editor and a resource compiler. For more advanced Macintosh programming Apple released several preliminary versions of MacApp, an object–oriented programming environment centered around the Object Pascal language. MacApp was based upon Apple's work with ToolKit/32, an ambitious object–oriented system based upon Clascal, the predecessor to Object Pascal. ToolKit/32 and Clascal were never supported by Apple since when both of these were near completion Apple diverted its resources toward the Macintosh and away from the Lisa.

EXEC FILES

The program development cycle for a typical application involves a lot of typing. To ease programmers of this burden the WorkShop allows command line input to come from special files, called Exec files. These files are similar to Apple's older /// Pascal Exec facility, but the Lisa version allows Exec files to be created with the editor. The Lisa Exec facility evolved into a very sophisticated Pascal–like language that supports variables, boolean expressions, loops, comments, and file I/O. A simple example follows which compiles two units & a main program, links these items, and runs the final executable file:

```
$EXEC
$
$SET %0 TO 'CARTOG/Map_Util'     { main program file }
$SET %1 TO 'CARTOG/UNewMapMgr'   { new map file manager unit }
$SET %2 TO 'CARTOG/UOldMapMgr'   { old map file manager unit }
$
P{ascal} %2                      { compile the old map manager unit }
         {}
         {}
P{ascal} %1                      { compile the new map manager unit }
         {}
         {}
P{ascal} %0                      { compile the main program }
         {}
         {}
$
L{ink}   %0                      { link the main program }
         %1                      { link the new map manager unit }
         %2                      { link the old map manager unit }
         Lisa/UVM                { link my virtual memory unit }
         Lisa/UDebug             { link my runtime debugger unit }
         IOSPasLib               { link the Lisa Pascal runtime unit }
         {}
         {}
         %0                      { send Linker output to the main program }
$
R{un}    %0  { run the main program }
$
$ENDEXEC  { That's all, Folks ... }
```

LANGUAGE UTILITIES

The WorkShop contains many useful utilities which assist programmers greatly. These utilities disassemble compiled programs, edit files at the block level, generate program cross–references, etc... The following table summarizes the more important utilities:

CodeSize	shows the code sizes and names of program segments
DumpObj	disassembles compiled program code
DumpPatch	edits files at the block level
FileDiv	divides large files into smaller files
FileJoin	joins files divided by FileDiv
Find	searches text files for strings
MacCom	transfers files to and from Lisa and Macintosh microdiskettes
PasMat	formats Pascal source files using many options
ProcNames	shows the names and lexical levels of Pascal program routines
RMaker	compiles a resource file for Macintosh programs
SegMap	shows the code segments in a program
ShowInterface	shows the interface portion of a Pascal Unit object file
UXRef	shows the routine relationships in a compiled program
XRef	shows the variable/routine cross-references in a source file

Two other features make the WorkShop well rounded. The **TransferProgram** command in the main command line runs a mouse– and window–based telecommunication's program. This is ideal for accessing a programming BBS and downloading source code. The **MakeBackground** command allows programs to be run as a background process. With this feature you can run several programs at once, but with several programs running concurrently system performance can become degraded.

CONCLUSION

For the past decade I have worked with many different computers, from programmable hand–held calculators to mainframes and from 1984 onward I have used a Lisa for programming mainly in Pascal. Overall, I have found the Lisa WorkShop to be a very powerful development system for creating, compiling, and debugging large programs. Even when compared to the other machines that I have worked with the Lisa WorkShop still comes out as being both a professional and powerful development environment.

<<< That's all, Folks ... >>>

WDS Extract Program

by William D. Smith

This program is used to extract words from Pascal source files. The words are written to a file for later processing (more on this next time). To use, enter names for the "input file," the "output file" (which defaults to "E.<input file>" after entering the input file) and the "skip words file" (which is optional). Any words in the "skip words file" are skipped when the input file is read. (My default "skip words file" has a list of all Pascal keywords and reserved words. This produces a list of all the words declared or used in the source file.) Words within comments and quotes are ignored.

The figure at the bottom of the page shows what you see as the program is running. "!" redisplays the screen, "E(xtract" extracts the words, "eX(amine" lets the you examine the files and "Q(uit" exits the program. "I", "O" and "S" let you change the file names. The at symbol "(@)" on the "input file" and "skip words file" lines allows you to enter the name of a file containing a list of file names which are processed instead of the named file by prefixing the file name with the at symbol.

The output file, shown on the next page, contains comment lines identifying the program, the date and time and which files are used and how. Then the number of words output (used in processing the file later) and finally the list of words, in alphabetical order, one per line.

The data structures used in the program are fairly simple. `Entries` is an array of records with two fields, `Skip` to tell if the word should be printed, which is set to `true` when words are read from the "skip words file(s)" and set to `false` when the words are read from the "input file(s)." The second field is an index into the `Strs` array (more on this later).

The grunt work in this program is done in the function `NextWord` which parses each line and returns one word at a time. When the end of a line is reached, `NextWord` gets the next line of the file. The function `Ins_Str` (called in the while loop of `ReadFile`) finds or inserts the word into the `Strs` array and returns the index of the word. No duplicates are inserted in the array.

The `Entries` array is then sorted alphabetically, ignoring case and underlines, and then printed. The quick sort routine described in the February 1989 NewsLetter is used for the sorting. The greater than or equal function, `Geq`, controls the sort order.

Now for some tricky stuff. The include file "`I.INS_STR.TEXT`" contains the `Ins_Str` function. A long time ago, I wrote an application (which is still in use) which needed to have in memory many names. Each name was stored on disk as `string [23]` as part of a complex record structure. I didn't have enough space to put all the names in memory at the same time. Statistics on the the names showed that the shortest name was 2 characters, the longest was 22 and the average was 11. 23 characters take up 12 words of memory while 11 characters take up only 6 words. There were also many duplicates (about 15 percent). Thus by removing duplicates and using an average of 6 words, I could get them all in memory. So what I needed was a data structure which was very space efficient in terms of storing strings.

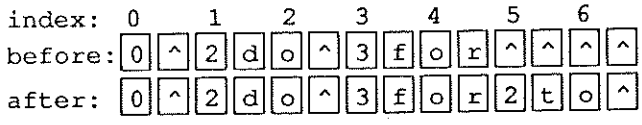
```
Extract: ! <options> E(xtract eX(amine Q(uit                    [1.05]
  Words:  137, Size:  480


  I = Input file (@) ---------> EXTRACT.TEXT

  O = Output file ------------> E.EXTRACT.TEXT

  S = Skip words file (@) ----> M22:WORDS.PAS.TEXT


Extracting from EXTRACT.TEXT
< 100>.........
 (c) CopyRight 1987 to 1990 by William D. Smith, All rights reserved
```

```
. Extract [1.05] on 05-09-90 at 09:43pm
.   Excludes words from M22:WORDS.PAS.TEXT
.   Includes words from WDS0:EXTRACT.TEXT
```

00170	G_NxtLn	OutName	T_Io_U
	G_Str	Oy	Ts
Abort	Height	P_Entries	Tt
AppendText	I	P_Ln	T_Tio_U
AtFile	I_into_S	Pop	Tx_Io_U
AtFiles_U	InCrlyCmt	ProcessFile	Tx_New
CapStr	Index	Prompt	Tx_Old
C_Eof	Indx	Px	UserAbort
C_Eol	InForm	Py	Vc_AtFiles_U
C_Eop	Initialize	Q_Sort	Vc_Display_U
Ch	InName	Q_Sort_U	Vc_Glbs_U
CharSet	InQuote	Qx	Vc_OpSys_U
Ck_Version	Ins_Str	ReadFile	Vc_Q_Sort_U
CloseAtFile	InStarCmt	S	Vc_StrOps_U
CloseText	InWord	SaveStr	Vc_T_Io_U
C_Msg	Iy	SetXy	Vc_T_Tio_U
CopyRight	J	Si	Vc_Tx_Io_U
Count	Junk	Skip	Version
C_Sc	Last	SkipName	Vs_AtFiles_U
Cursor	Len	Skipped	Vs_Display_U
D	Line	SkipThese	Vs_Glbs_U
Delete_Ul	LoadWords	S_Menu	Vs_OpSys_U
D_into_S	MaxEntry	S_Misc	Vs_Q_Sort_U
DisplayFile	MaxStr	S_Msg	Vs_StrOps_U
Display_U	M_Eof	So	Vs_T_Io_U
Done	MinEntry	S_Prompt	Vs_T_Tio_U
Entries	MinStr	Src	Vs_Tx_Io_U
Entry	M_NoError	SrcS	Vv_AtFiles_U
Err	Msg	Str_1	Vv_Display_U
Err_Msg	Name	Str_15	Vv_Glbs_U
Error	NextAtFile	Str_23	Vv_OpSys_U
Esc	NextStr	Str_255	Vv_Q_Sort_U
Examine	NextWord	Str_31	Vv_StrOps_U
Extract	Nm	Str_63	Vv_T_Io_U
ExtractWords	N_Menu	StrOps_U	Vv_T_Tio_U
F	Null	Strs	Vv_Tx_Io_U
FibPtr	Numbers	Sy	W_Dot
Filename	Off	T	Word
G_Char	OkChars	Tad	W_Str
Geq	OpenAtFile	TadRec	X
Get_Name	OpenText	Tgt	Xx
Get_Sys_Tad	OpSys_U	TgtS	Y
Glbs_U	OutFile	T_into_S	Yy
Cont. next column	*Cont. next column*	*Cont. next column*	

Knowing something about how the compiler and strings interacted, I devised the following scheme: any program which needed to store variable length strings in memory would declare an array of `string [1]`. This indicates that each word in the array is a string of length one. (The first byte of a string is the length and strings always use an even number of bytes of storage although the last byte may not be used.) `Ins_Str` returns the index of string in the array. To do this, it searches through the array and finds the string in which case the index is returned, or does not find it, in which case the string is inserted and the index where the string was inserted is returned. Any index not returned by Ins_Str is not allowed (eg. 2 in the following example) because a legal string does not start at that point. The array is used by naming it with an index (eg. `Strs [Nm]` in `P_Entries`). This technique works because all strings are type compatible. By convention the first string in the array is always the empty string.

In the following example, `Ins_Str` returns 3 when called with "for" and returns 5 when called with "to". The string "to" is inserted into the array at index 5. 2, 4 & 6 are illegal indices. "^"s are unused bytes. The index is by words.



`Ins_Str` uses a modified Boyer-Moore search. The first word of the string to be inserted (the length byte and the first character) are compared to the first string. If they don't match, the index into the array is incremented by the length of the string. If they match, the string is then compared to the string in the array at the index location. Again, if they don't match the string is incremented by the length of the string in the array. This is repeated until a match is found or the unused part is found. New strings are always inserted at the end. For reasons of speed, part of `Ins_Str` is coded using the `pmachine` instruction.

```
{ Extract words from pascal source [1.05] --- 30 Mar 88 }  { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1987 to 1990,  All rights reserved.              }

{ File:          Extract.Text              Version 1.05    30 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) 1987 to 1990.

  System:        Power System version IV.2.2

  Compiler:      Power System Pascal Compiler

  Keywords:      WDS Extract Program Pascal

  Description:   This program extracts all the words from the input file.
                 Lines begining with a period in the "skip words file" are
                 ignored.  Words in comments and quotes are skipped in the
                 "extract from file".  It then prints out all those words
                 which are not in the "skip words file".

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
29 Mar 88   WDS  1.04  Changed to use new WDS units.
06 Feb 88   WDS  1.03  Changed to reflect changes to T_Io_U for C_Sc.
24 Jan 88   WDS  1.02  Fixed two minor errors.
23 Oct 87   WDS  1.01  Finished initial program, now start testing.
15 Jan 87   WDS  1.00  Started.
}

{$I VERSION.TEXT}  { Declares conditional compilation flags }

program Extract;

uses  Glbs_U,      { WDS globals unit }
      StrOps_U,    { WDS string ops unit }
      Tx_Io_U,     { WDS text 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 }
      AtFiles_U,   { WDS "@" file unit }
      Q_Sort_U;    { WDS quicksort unit }
```

```
const Version    = '[1.05]';    { 30 Mar 88 }
      MinEntry   = 0;
      MaxEntry   = 2000;

      MinStr     = 0;
      MaxStr     = 10000;

type  Entry      = packed record
                     Skip      : boolean;      { On output, skip this word }
                     Nm        : 0..MaxStr;    { Index into Strs }
                   end { record };

var   Index      : integer;       { Next usable entry in Entries }
      NextStr    : integer;       { Next usable word in Strs }

      Ch         : char;
      N_Menu     : boolean;       { Need the menu displayed }

      Iy         : integer;       { Input file line }
      Oy         : integer;       { Output file line }
      Px         : integer;       { Prompt column }
      Qx         : integer;       { Question column }
      Sy         : integer;       { Skip words line }
      I          : integer;

      InName     : Str_31;        { Need room for '@' }
      SkipName   : Str_31;        { Need room for '@' }
      OutName    : Str_31;

      OkChars    : CharSet;       { Set of legal characters }
      Numbers    : CharSet;

      Entries    : array [MinEntry..MaxEntry] of Entry;
      Strs       : array [MinStr..MaxStr] of Str_1;

  procedure Initialize;
  var I : integer;  S : Str_15;
  begin
    S := 'Extract';

    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_Tx_Io_U, Vc_Tx_Io_U, S, Vs_Tx_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);
    Ck_Version (Vv_AtFiles_U, Vc_AtFiles_U, S, Vs_AtFiles_U);
    Ck_Version (Vv_Q_Sort_U, Vc_Q_Sort_U, S, Vs_Q_Sort_U);

    OkChars := ['A'..'Z', 'a'..'z'];
    Numbers := ['_', '0'..'9'];

    fillchar (Entries, sizeof (Entries), 0);
(*
    for I := MinEntry to MaxEntry do begin
      with Entries [I] do begin
        Skip := false;
        Nm := MinStr;
      end { with };
    end { for };
*)
    Index := MinEntry;
```

```
  fillchar (Strs, sizeof (Strs), 0);
  NextStr := MinStr + 1;

  N_Menu := true;

  InName := '';
  OutName := '';
  SkipName := 'M22:WORDS.PAS.TEXT';

  S_Msg (false, false, CopyRight);
end { Initialize };

procedure S_Misc (Index, NextStr : integer);
var S : Str_31;
begin
  SetXy (3, Py + 1);
  S := 'Words:    0, Size:    0';
  I_into_S (Index, S, 8, 4);
  I_into_S (NextStr, S, length (S) - 3, 4);
  W_Str (S);
end { S_Misc };

procedure S_Menu (Prompt : boolean);
begin
  if Prompt then
    begin
      C_Sc (false);
      Px := S_Prompt ('Extract: ! <options> E(xtract eX(amine Q(uit',
                      Version, false);
      S_Misc (Index, NextStr);
    end { if };

  SetXy (3, Py + 5);  Iy := Y;
  W_Str ('I = Input file (@) ---------> ');  W_Str (InName);

  SetXy (3, Y + 2);  Oy := Y;
  W_Str ('O = Output file ------------> ');  W_Str (OutName);

  SetXy (3, Y + 2);  Sy := Y;
  W_Str ('S = Skip words file (@) ----> ');  Qx := X;  W_Str (SkipName);

  N_Menu := false;
end { S_Menu };

function Get_Name (var S : Str_23;  Xx, Yy, Len : integer) : boolean;
begin
  Get_Name := false;
  SetXy (Xx, Yy);

  if G_Str (S, Len) <> chr (Esc) then
    begin
      AppendText (S, true);
      SetXy (Xx, Yy);  W_Str (S);
      C_Eof (Len - length (S));
      Get_Name := true;
    end { if };
end { Get_Name };

procedure Delete_Ul (var Si : Str_255;  var So : Str_255);
var I, J : integer;
begin
  J := 0;
```

```
    for I := 1 to length (Si) do begin
      if Si [I] <> '_' then
        begin
          J := J + 1;
          So [J] := Si [I];
        end { if };
    end { for };

    So [0] := chr (J);
    CapStr (So);
  end { Delete_Ul };

  function Geq (var Src, Tgt : integer { Entry }) : boolean;
  var Ts, Tt : ^Entry;  SrcS, TgtS : Str_255;
  begin
    pmachine (^Ts, ^Src, 196 { Sto });  { Ts := Src; }
    pmachine (^Tt, ^Tgt, 196 { Sto });  { Tt := Tgt; }

    Delete_Ul (Strs [Ts^ .Nm], SrcS);
    Delete_Ul (Strs [Tt^ .Nm], TgtS);

    Geq := SrcS >= TgtS;
  end { Geq };

{$I I.INS_STR.TEXT}

  procedure ExtractWords;
  label 2;
  var   Err     : boolean;
        Msg     : integer;
        Skipped : integer;
        Tad     : TadRec;
        OutFile : FibPtr;
        Junk    : Entry;
        S       : Str_63;

    function ReadFile (Name : Str_23;  SkipThese : boolean) : boolean;
    { This function reads the file, seperates each line into words,
      and inserts the words into the Strs array.  Msg is the message returned
      from G_NxtLn or M_NoError if G_NxtLn not called.
    }
    label 1;
    var   F       : FibPtr;
          Abort   : boolean;
          Count   : integer;
          I       : integer;
          Msg     : integer;
          SaveStr : integer;
          Indx    : integer;    { Next char in the input line }
          Line    : Str_255;    { Current input line }
          Word    : Str_255;

      function NextWord (var S : Str_255) : boolean;
      label 1;
      var   Done      : boolean;
            I         : integer;  { Start of the word }
            Msg       : integer;
            InCrlyCmt : boolean;
            InStarCmt : boolean;
            InQuote   : boolean;
            InWord    : boolean;
      begin
```

```
NextWord := false;
S [0] := chr (0);  { S := ''; }
InStarCmt := false;
InCrlyCmt := false;
InWord := false;

repeat
  while Indx > length (Line) do begin
    if UserAbort then
      begin  Abort := true;  goto 1;  end { if };

    W_Dot (Count);

    if not G_NxtLn (F, Line, Msg) then
      begin
        if Msg <> M_Eof then
          begin  Err_Msg (Msg, Name);  Abort := true;  end { if };

        goto 1;
      end { if };

    InQuote := false;  { Quotes are limited to same line }
    Indx := 1;

    if SkipThese then
      if length (Line) > 0 then
        if Line [1] = '.' then
          Indx := length (Line) + 1;
  end { while };

  Done := false;

  repeat
    if Indx > length (Line) then  Done := true
    else if InWord then
      if Line [Indx] in OkChars then
      else if Line [Indx] in Numbers then
      else  Done := true
    else if InCrlyCmt then
      if Line [Indx] = '}' then  InCrlyCmt := false
      else
    else if InQuote then
      if Line [Indx] = '''' then  InQuote := false
      else
    else if InStarCmt then
      begin
        if Line [Indx] = '*' then
          if Indx >= length (Line) then  Done := true
          else if Line [Indx + 1] = ')' then
            begin
              InStarCmt := false;
              Indx := Indx + 1;
            end { if };
      end { else if }
    else if Line [Indx] in OkChars then
      begin
        I := Indx;
        InWord := true;
      end { if }
    else if not SkipThese then
      if Line [Indx] = '{' then  InCrlyCmt := true
      else if Line [Indx] = '''' then  InQuote := true
      else if Line [Indx] = '(' then
        if Indx >= length (Line) then  Done := true
        else if Line [Indx + 1] = '*' then
```

```
                begin
                  InStarCmt := true;
                  Indx := Indx + 1;
                end { if };

          Indx := Indx + 1;
        until Done;

        if InWord then
          S := copy (Line, I, Indx - I - 1)
      until InWord;

      NextWord := true;
    1:
    end { NextWord };

  begin { ReadFile }
    ReadFile := false;

    if not OpenText (F, Name, Tx_Old, Msg) then
      begin  Err_Msg (Msg, Name);  goto 1;  end { if };

    Msg := M_NoError;
    Abort := false;
    Count := 0;
    Line := '';
    Indx := length (Line) + 1;

    while NextWord (Word) do begin
      SaveStr := NextStr;
      I := Ins_Str (Strs, Word, NextStr);

      if NextStr = Null then
        begin
          Error ('No more room to store the name');
          goto 1;
        end { if }
      else if SaveStr <> NextStr then  { new word, NextStr was increased }
        begin
          if Index >= MaxEntry then
            begin
              Error ('No more room in entry table');
              goto 1;
            end { if }
          else
            begin
              with Entries [Index] do begin
                Skip := SkipThese;
                Nm := I;
              end { with };

              Index := Index + 1;
            end { else };
        end { else if };
    end { while };

    ReadFile := not Abort;
  1:
    CloseText (F, false);
  end { ReadFile };

  function LoadWords (Name : Str_23;  SkipThese : boolean) : boolean;
  var F : FibPtr;  Msg : integer;  S : Str_31;
  begin
    LoadWords := false;
```

```
  if SkipThese then InForm ('Loading from ', Null, Name)
  else  InForm ('Extracting from ', Null, Name);

  if ReadFile (Name, SkipThese) then
          {12345678 1 2345678 3 234}
      S := '.   Includes words from ';

      if SkipThese then
        begin  S [5] := 'E';  S [6] := 'x';  end { if };

      if P_Ln (OutFile, concat (S, Name), 1, Msg) then  LoadWords := true
      else  Err_Msg (Msg, OutName);
    end { if };

  S_Misc (Index, NextStr);
end { LoadWords };

function ProcessFile (Filename : Str_31;  SkipThese : boolean) : boolean;
{ Opens and processes the file(s) }
label 2;
var   AtFile : FibPtr;
      Err    : boolean;
      Msg    : integer;
      S      : Str_23;
begin
  Err := false;

  if OpenAtFile (AtFile, Filename, Msg) then
    begin
      while NextAtFile (AtFile, S, Msg) do begin
        CapStr (S);

        if not LoadWords (S, SkipThese) then
          begin  Err := true;  goto 2;  end { if };
      end { while };

      if Msg <> M_Eof then
        begin  Err_Msg (Msg, Filename);  Err := true;  end { if };
    2:
      CloseAtFile (AtFile);
    end { if }
  else if Msg <> M_NoError then
    begin  Err_Msg (Msg, Filename);  Err := true;  end { else if }
  else  Err := not LoadWords (Filename, SkipThese);

  ProcessFile := not Err;
end { ProcessFile };

procedure P_Entries (Last : integer);
label 1;
var   Count, I : integer;  Msg : integer;
begin
  Count := 0;

  for I := MinEntry to Last do begin
    with Entries [I] do begin
      if not Skip then
        begin
          W_dot (Count);

          if not P_Ln (OutFile, Strs [Nm], 1, Msg) then
            begin  Err_Msg (Msg, OutName);  goto 1;  end { if };
        end { if };
    end { with };
  end { for };
```

```
    1:
    end { P_Entries };

  begin { ExtractWords }
    Cursor (Off);

    if OpenText (OutFile, OutName, Tx_New, Msg) then
                                              {2 2 8765432 1 876543210}
        S := concat ('. Extract ', Version, ' on 00-00-00 at 00:00am');
        Get_Sys_Tad (Tad);
        D_into_S (Tad .D, S, length (S) - 18);
        T_into_S (Tad .T, true, S, length (S) - 6);

        if not P_Ln (OutFile, S, 1, Msg) then  goto 2;

        Skipped := 0;
        Index := MinEntry;
        NextStr := MinStr + 1;

        if not ProcessFile (SkipName, true) then
          begin  Err := true;  goto 2;  end { if };

        Skipped := Index;

        if not ProcessFile (InName, false) then
          begin  Err := true;  goto 2;  end { if };

        if Index > MinEntry then
          begin
            Inform ('Sorting ', Index, ' words');
            SetXy (0, Height - 1);
            C_Eol;
            Q_Sort (Entries, Junk, MinEntry, Index - 1, sizeof (Junk), Geq);

            if not P_Ln (OutFile, '', 1, Msg) then  goto 2;

            S := '00000';
            I_into_S (Index - Skipped, S, 1, 5);

            if not P_Ln (OutFile, S, 2, Msg) then  goto 2;

            Inform ('Printing ', Index - Skipped, ' words');
            P_Entries (Index - 1);
          end { if };
      2:
        CloseText (OutFile, not Err);
        Inform ('', Null, '');
        S_Misc (Index, NextStr);
      end { if };

    if Msg <> M_NoError then  Err_Msg (Msg, OutName);

    Cursor (Pop);
  end { ExtractWords };

  procedure Examine;
  var Ch : char;  Xx : integer;  S : Str_31;
  begin
    Py := Py + 1;
    Xx := S_Prompt ('Examine: ! I(nput O(utput S(kip Q(uit', Version, false);

    repeat
      if N_Menu then  S_Menu (false);

      S [0] := chr (0);  { S := ''; }

      SetXy (Xx, Py);
      Ch := G_Char (['!', 'I', 'O', 'Q', 'S'], false, true);
```

```
      case Ch of
        '!' : begin
                SetXy (0, Py + 1);
                C_Eop (false);
                N_Menu := true;
              end { case '!' };
        'I' : S := InName;
        'O' : S := OutName;
        'S' : S := SkipName;
      end { cases };

      if length (S) > 0 then
        begin
          if S [1] = '@' then  delete (S, 1, 1);

          if DisplayFile (Py + 1, Null, S) then  N_Menu := true;
        end { if };
    until Ch = 'Q';

    SetXy (0, Py);
    C_Eol;
    Py := Py - 1;
    S_Misc (Index, NextStr);
  end { Examine };

begin { Extract }
  Initialize;
  S_Menu (true);
  Ch := 'I';

  repeat
    case Ch of
      '!' : N_Menu := true;

      'E' : ExtractWords;

      'I' : begin
              if Get_Name (InName, Qx, Iy, 24) then
                if length (OutName) = 0 then
                  if length (InName) > 0 then
                    if length (InName) <= 21 then
                      begin
                        OutName := InName;

                        if OutName [1] = '@' then  delete (OutName, 1, 1);

                        I := pos (':', OutName);

                        if I = 0 then
                          if InName [1] = '*' then  I := 1;

                        insert ('E.', OutName, I + 1);
                        SetXy (Qx, Oy);
                        W_Str (OutName);
                      end { if if if if };
            end { case 'I' };

      'O' : if Get_Name (OutName, Qx, Oy, 23) then ;

      'Q' : { nothing };

      'S' : if Get_Name (SkipName, Qx, Sy, 24) then ;

      'X' : Examine;
    end { cases };

    if N_Menu then  S_Menu (true);
```

```
    SetXy (Px, Py);
    Ch := G_Char (['!', 'E', 'I', 'O', 'Q', 'S', 'X'], false, true);
  until Ch = 'Q';

  C_Msg (0);
end {$Q- Extract }.
```

The conditional compiler directives "BYTE" and "INS_STR_FWD" are declared in the include file "VERSION.TEXT". "BYTE" is true for byte addressed machines and false for word addressed machines. "INS_STR_FWD" is redeclared in the program including the file if Ins_Str must be declared forward. This include file should actually be turned into a unit, but I haven't had time to do that. I've used it successfully in 15 to 20 different programs on several different machines and versions of the p-System. I stripped out some code for different pmachine versions to save space.

```
{ Insert a str in an array of strs [1.00] --- 23 Oct 87 }  { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1987 to 1990,  All rights reserved.              }

{ File:          I.Ins_Str.Text          Version 1.00    23 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 1990.

  System:        Power System version IV.2.2

  Compiler:      Power System Pascal Compiler

  Keywords:      WDS Insert String Include

  Description:   Insert string include file.  This file contains a function for
                 inserting strings in an array of Str_1.  If there is no more
                 room to add a name into the array A, Ins_Str is returned as
                 MinStr and N (the index of the next unused string in the array)
                 is set to Null.  Prior to use, N must be set to MinStr + 1
                 and the first word of A must be filled with Nuls (0).  S is the
                 string to be inserted and is unchanged.

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
23 Oct 87   WDS  1.00  Written based on I.Ins_Name.
}

{$B INS_STR_FWD-}
    function Ins_Str (var A : interface
                                array [MinStr..MaxStr : integer] of Str_1;
                      var S : Str_255;
                      var N : integer) : integer;
{$E INS_STR_FWD-}

{$B INS_STR_FWD+}
    function Ins_Str { (var A : interface
                                  array [MinStr..MaxStr : integer] of Str_1;
                        var S : Str_255;
                        var N : integer) : integer };
{$E INS_STR_FWD+}
```

```
    label 1;

    const Sind0 = 120;  { Short load indirect }
          Adi   = 162;  { Add integers }
          Sto   = 196;  { Store indirect }
          Dup1  = 226;  { Duplicate 1 word at top of stack }
          Efj   = 210;  { Equal false jump }
          Fjp   = 212;  { False jump }
          Ixa   = 215;  { Index array }

    var   Base  : integer;
          TmpI  : integer;
          TmpN  : integer;
          Words : integer;
          Ai    : ^Str_1;       { Address of A [I] }
    begin
      TmpI := MinStr;

      if length (S) > 0 then
        begin
          if N <> Null then  TmpN := N
          else  TmpN := MaxStr;

          pmachine (^Base, ^A, Sto,      { Base := address (A); }
                    ^S, Sind0);          { Load value of S [0], S [1] }

          while TmpI < TmpN do begin
            pmachine (^Ai,               { Ai ... }
                      (Base), (TmpI),
{$B BYTE+}
                      Dup1, Adi,         { TmpI * 2 on byte address machine }
{$E BYTE+}
                      Adi, Sto,              {... := Base + I; }
                      Dup1,              { Duplicate S [0], S [1] }
                      (Ai), Sind0,       { Load value of A [I] }
                      Efj,  9);          { EFJ to INCR:, if <> jump to INCR: }
            if Ai^ = S { compare strings } then  goto 1;  { match found }

  { INCR: } TmpI := TmpI + ((length (Ai^) + 2) div 2);
          end { while };

          Words := (length (S) + 2) div 2;

          if TmpN + Words <= MaxStr then
            begin
{$R-} { In case range checking not already off }
              moveleft (S [0], A [TmpN], length (S) + 1);
{$R^} { Pop back the original value of range checking }
              TmpI := TmpN;
              N := TmpN + Words;
            end { if }
          else
            begin
              TmpI := MinStr;
              N := Null;
            end { else };
        1:
          pmachine (Fjp, 0);  { Use up one word on the stack }
        end { if };

      Ins_Str := TmpI;
    end { Ins_Str };
```

```
{ WDS "@" files unit                    [1.00] --- 29 Mar 88 }  { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1988 to 1990,  All rights reserved.                  }
{ File:          AtFiles_U.Text          Version 1.00    29 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) 1987 to 1990.

  System:        Power System version IV.2.2

  Compiler:      Power System Pascal Compiler

  Keywords:      WDS AtFiles_U At Files Unit

  Description:   This unit contains procedures which process "@" files.

  Change log: (most recent first)

Date        Id   Vers  Comment
---------   ---  ----  -------
29 Mar 88   WDS  1.00  Extracted from Extract and made a unit.
}

{$I VERSION.TEXT}  { Declares conditional compilation flags }

unit  AtFiles_U;

interface  {$ AtFiles_U [1.00] 29 Mar 88 }

uses  Glbs_U;    { WDS globals unit }

const Vc_AtFiles_U = 1;  { 29 Mar 88 }
      Vs_AtFiles_U = 'AtFiles_U';

var   Vv_AtFiles_U : integer;

  function OpenAtFile (var AtFile   : FibPtr;
                           AtName   : Str_31;
                       var Msg      : integer) : boolean;
  { Open "@" file.  This function returns true only if "AtName" begins with
    an "@" sign and the file following the "@" sign is opened.  If file is
    not opened, "Msg" contains the ioresult.
  }

  procedure CloseAtFile (var AtFile : FibPtr);
  { Close "@" file.  This procedure closes the "@" file if it is not closed. }

  function NextAtFile (var AtFile : FibPtr;
                       var S      : Str_23;
                       var Msg    : integer) : boolean;
  { Next "@" file.  This function returns the name of the next file in the
    "@" file in "S".  If there are no more files or there is an error reading
    the "@" file, "S" is returned empty.  Msg is returned as "M_NoError",
    "M_Eof", or the ioresult.  The function returns true only if "Msg" is
    "M_NoError".  No empty lines and no lines begining with a "." are
    returned in "S".  All spaces are removed and the line is always truncated
    to 23 characters (maximum length of a filename).
  }
```

```
implementation

uses  StrOps_U, { WDS string ops unit }
      Tx_Io_U;  { WDS text I/O unit }

  function OpenAtFile { (var AtFile   : FibPtr;
                             AtName   : Str_31;
                         var Msg      : integer) : boolean };
  begin
    OpenAtFile := false;
    AtFile := Closed;
    Msg := M_NoError;

    if length (AtName) > 0 then
      if AtName [1] = '@' then
        begin
          delete (AtName, 1, 1);
          OpenAtFile := OpenText (AtFile, AtName, Tx_Old, Msg);
        end { if if };
  end { OpenAtFile };

  procedure CloseAtFile { (var AtFile : FibPtr) };
  begin
    if AtFile <> Closed then  CloseText (AtFile, false);
  end { CloseAtFile };

  function NextAtFile { (var AtFile : FibPtr;
                         var S      : Str_23;
                         var Msg    : integer) : boolean };
  var Done : boolean;  Lstr : Str_255;
  begin
    S [0] := chr (0);  { S := ''; }
    Done := false;

    repeat
      if G_NxtLn (AtFile, Lstr, Msg) then
        begin
          if length (Lstr) > 0 then
            if Lstr [1] <> '.' then
              begin
                Crunch_Str (Lstr, Lstr);

                if length (Lstr) > 23 then  Lstr [0] := chr (23);

                S := Lstr;
                NextAtFile := true;
                Done := true;
              end { if if };
        end { if }
      else  Done := true;
    until Done;

    NextAtFile := Msg = M_NoError;
  end { NextAtFile };

begin { AtFiles_U }
  Vv_AtFiles_U := Vc_AtFiles_U;

  Ck_Version (Vv_Glbs_U, Vc_Glbs_U, Vs_AtFiles_U, Vs_Glbs_U);
  Ck_Version (Vv_StrOps_U, Vc_StrOps_U, Vs_AtFiles_U, Vs_StrOps_U);
  Ck_Version (Vv_Tx_Io_U, Vc_Tx_Io_U, Vs_AtFiles_U, Vs_Tx_Io_U);

  *** ;

end {$Q- AtFiles_U }.
```

Board Meeting Minutes (April 11, 1990)
By Keith R. Frederick

Minutes of the Board Meeting of USUS, Inc., held in room 1 of the MUSUS forum teleconferencing facility on the CompuServe Information Service April 11, 1990.

Present at the meeting were:

User ID	Name
74076,1715	Felix Bearden (Felix) with **Bob Spitzer (BobS) BoD**
76702,513	Harry Baya (Harry)
73447,2754	**Henry Baumgarten (Henry) BoD**
72767,622	**Tom Cattrall (TomC) BoD**
73760,3521	Keith Frederick (Keith)
72747,3126	Bob Clark (BobC)

Topics dealt with were (primarily based on a proposal by Hays):

TRADING THE SAGE IV

The first item on the agenda was a proposal by Hays suggesting that USUS trade the Sage IV for peripherals that would be used on the Sage II, which was deemed adequate for the administrative duties that would be delegated to it. Since Felix Bearden is the new administrator, Henry, Tom, and Harry echoed that if Felix didn't mind they would vote for the trade. Felix mentioned that it didn't directly effect him since he would use his 460 anyway, but that future administrators would need the peripherals. So, the motion to trade the Save IV for peripherals for the Sage II was voted upon and unanimously approved. Felix stated he would implement the motion.

ADVERTISING IN JPAM

Felix suggested that as soon as the JPAM agreement was implemented USUS should place an ad in JPAM. The price would be about $1100 for a one-third page ad.

Bob Clark, Treasurer, noted the ad price is about one-fifth of the current bank balance of USUS and stated the account is slowly draining away. Tom Cattrall felt the advertisement is too much considering the present state of USUS and suggested looking for lower cost alternatives or just waiting. Bob Spitzer suggested that since USUS has a bad case of the dwindles sitting tight won't do anything to help and we need to invest in a shot of growth. Also pointing out that given JPAM's circulation of over 15,000, even a tiny response would be dramatic.

Bringing up a related topic, Bob Clark asked about the plan to have a USUS column in JPAM. Bob Spitzer commented that the column issue is up in the air and getting an author is an important limiting factor. Tom Cattrall suggested that the idea of an article in JPAM coupled with upcoming MUSUS publicity would be better considering the resources of USUS at the moment. Further stating that our best bet would be to start with an article or two in JPAM that got USUS in the public eye.

However, Bob Spitzer, reminded everyone that if an article was submitted this summer it would not get reviewed, revised, and published until a year from the submission. Having a regular column would be just as bad and that the advertisement would be needed to tell everyone about USUS. Also, it would only take twenty new trial memberships to pay back the $1100.

Henry stated that his calculation is a bit more conservative, estimating forty or fifty new memberships would be needed to pay for the advertisement. Tom Cattrall commented that even Henry's number is not enough given USUS also gives new members many benefits that cost money. Tom also stated that the JPAM editorial cycle is currently quite fast, although some delay would still occur.

Bob Clark asked if we got any benefit from joining with JPAM. Felix answered stating that our affiliation will build with JPAM, but for now at least USUS can say join us and you get JPAM as part of the package. Felix then asked if anyone had ideas on letting the world know about USUS. Tom restated his wish to look at alternatives to the $1100 advertisement and suggested that we discuss this on MUSUS in the coming days.

Bob Spitzer replied that his experience in the computer world showed that if you don't advertise, you aren't seen. Tom responded that USUS should at least consider all options before deciding on the solution.

Henry proposed a compromise: prepare an ad and we can decide whether or not it would be likely to have the desired affects. Henry also indicated that he tends to favor the ad. Felix and Bob Spitzer said they would give the writing of the text for the advertisement a try and Harry mentioned he would like to take a shot at the artwork.

BOARD MEMBER PARTICIPATION

A proposal was made to try and remove board members who don't participate in the meetings. Since the current bylaws don't provide for such an action, Felix stated that he would write a motion to revise the bylaws at the next meeting. Tom Cattrall suggested that non-participating board members, indicating Frank Lawyer and Stephen Pickett as such, be asked to participate or resign. Also noting that he would rather have them show up than resign.

Felix suggested that a change in the bylaws would be the least painful way of taking care of the situation. Harry thought

though, that the problem is not too great in that if a board member was asked to resign he probably would. Also, Harry stated that he has talked to Frank Lawyer recently and Frank was very busy and would probably accept a request to attend or resign.

Henry mentioned that Frank should be given a chance to decide but Stephen Pickett asked for the task and has only showed up for one meeting. Henry said he should be reminded or removed. However, Tom Cattrall realized that since Stephen Pickett had not been on MUSUS in some time he might not know he was elected. No one notified him and the newsletter, which mentions the election result, hadn't been mailed yet.

The topic of what to do was tabled until the next meeting.

[Minute Taker's Note : Tom Cattrall was correct; Stephen Pickett was not notified of his victory in the election and did not know he was elected.]

APPLE II STUFF

Hays had mentioned that he would safeguard the Apple IIe and master copies of the software library until USUS decides on what to do with it. Harry indicated that Frank Lawyer, current leader of the Apple II SIG, should be asked if we wants the Apple //e and if he would attend future board meetings.

Keith Frederick indicated that if the Apple II section needed someone to support it, he would volunteer. Henry mentioned that Frank has done an excellent job in the past and would be asked first he wants to continue heading the Apple II SIG.

NEXT MEETING

The Board agreed to adjourn and meet again at 6:30 PM PST / 7:30 MST / 8:30 CST / 9:30 EST May 9th 1990 in Room 1 of the MUSUS conference facility.

Minutes submitted by: Keith R. Frederick

Treasurer's Report
by Robert E. Clark, Treasurer

March 1990

Bank Balance	$6,471.02	2-28-90
Income - March 1990		
Dues:		(new/renew)
Student	0.00	n/a
General	35.00	n/a
Professional	0.00	n/a
Institutional	0.00	n/a
Other Income:		
CIS (2 checks)	86.74	
Library fees	0.00	
Publications	0.00	
PowerTools	180.85	
Total Income:	$ 302.59	
Expenses - March 1990		
Administrator:		
CIS	0.00	
Telephone	0.00	
Postage	6.29	
Photocopies	0.00	
Supplies	0.00	
Other:		
Bank charges	2.00	
Newsletter	0.00	
Mail from La Jolla	2.65	
La Jolla Box (6mo)	36.00	
Reimbursements	0.00	
Total Expenses	$ 46.94	
Bank Balance	$6,726.67	3-31-90

April 1990

Bank Balance	$6,726.67	3-31-90
Income - April 1990		
Dues:		(new/renew)
Student	0.00	n/a
General	285.00	n/a
Professional	100.00	n/a
Institutional	0.00	n/a
Other Income:		
CIS	27.82	
Library fees	0.00	
Publications	0.00	
PowerTools	0.00	
Total Income:	$ 412.82	
Expenses - April 1990		
Administrator: (Busch final)		
CIS	34.41	
Telephone	9.50	
Postage	105.41	
Photocopies	3.60	
Supplies	0.00	
Other:		
Bank charges	2.00	
Newsletter	0.00	
Mail from La Jolla	0.00	
Reimbursements	91.06	
Total Expenses	$ 245.98	
Bank Balance	$6,893.51	4-30-90

From The Editor
by Tom Cattrall

This is probably the biggest issue yet of the USUS newsletter. The backlog of articles is just about gone now. So, if you have an idea for an article, a review, or a letter, now is the time to send it in. Short items are especially welcome. With only long articles, it is difficult to make the size come out to the proper number of pages.

Sanyo PC Hackers Newsletter

In the last issue I forgot to mention that the inspiration for the review of the Impact print head repair service came from the Sanyo PC Hackers Newsletter. They had some discussions of trying to find replacement print heads and it occurred to me that I should write up my experiences. When the article came out, Victor Frank, the editor of the newsletter, called me to ask permission to reprint the article.

He included a word about USUS in the introduction and I feel it only fair that I mention his group as well. It covers the orphaned Sanyo 16 bit computers and provides a newsletter and a large (over 250 volumes) disk library. It looks like a good resource for anyone with a Sanyo computer.

Victor R. Frank
12450 Skyline Blvd.
Woodside, CA
94062-4541

Dues: $20/yr US, C$25 Canada, $25 surface mail elsewhere, $35 foreign airmail

Victor also mentioned that he has a Stride but hasn't done much with it yet. He said that some beginner's level articles on the Stride and the P-system would interest him. Is there anyone out there that has recently gone through the learning curve? If so, how about writing up some of your observations?

USUS board meeting minutes

There has been some discussion on MUSUS about how to report the board meeting minutes in the USUS newsletter. My position has been that people don't join USUS to read meeting minutes, and therefore I advocate a "minimalist" approach. Provide just enough to give people an idea of what is going on but don't get into lots of detail.

The danger in this is that some positions may not be adequately represented. An example is in this issue's minutes. During the discussion of board members that weren't attending meetings, I said that having only 3 board members out of 5 as regular participants made getting a quorum difficult. So I suggested that either the 2 missing members start attending or resign so that we would only need 50% of 3 rather than 50% of 5 in order to conduct business. In the minutes, my discussion of the quorum difficulties was not included. Thus, the reasons why we discussed "getting rid of" board members might be misinterpreted.

In spite of these possible misunderstandings, I still feel that small is better regarding minutes. But, I figured I'd bring it up in case anyone would like to lobby for another policy. If there is any meeting for which you wish to have more detail, contact me or another of the attendees and we could provide you with a complete transcript.

Submission Guidelines

Submit articles to me at the address shown on the back cover. Electronic mail is probably best, disks next best, and paper copy is last. If your article has figures or diagrams, I can use encapsulated Postscript files in any of the disk formats listed below. If you can't produce encapsulated Postscript, then paper copy is probably the only practical method for submitting graphics.

You can send E-Mail to my Compuserve ID: 72767,622, or indirectly from internet: 72767.622@compuserve.com. For disks, I can read Sage/Stride/Pinnacle format disks. Also, any MS-DOS 5.25 or 3.5 disks, and 3.5" Amiga disks. If anyone wants to send Mac format disks I could probably get someone to translate them into something I can use. Whatever you send, please mark on the disk what format it is. That will save me a lot of guesswork.
Text should be plain ascii rather than a word processor file. It can have carriage returns at the end of all lines or only at the ends of paragraphs. What you send doesn't have to look pretty. I will take care of that. My spelling checker will take care of spelling errors too. If you want special formatting use the following conventions:

1. _Underline_, put an underline character at each end of the section to underline.

2. *Bold*, put a star at each end of the section to **bold**.

3. ^Italics^, put a caret at each end of the section to be set in *italics*.

4. ??Special requests??, such as ??box next paragraph?? should be surrounded with "?? ??".

NewsLetter Editor : Tom Cattrall
Amity Software Inc.
7600 Seawood Road SE
Amity, Oregon 97101
503/835-1613
Compuserve : 72767,622
Internet : 72767.622@compuserve.com

NewsLetter Publisher : Robert H. Geeslin Ed.D.

USUS Membership Information

Student Membership	$ 30 / year
Regular Membership	$ 45 / year
Professional Membership	$ 100 / year

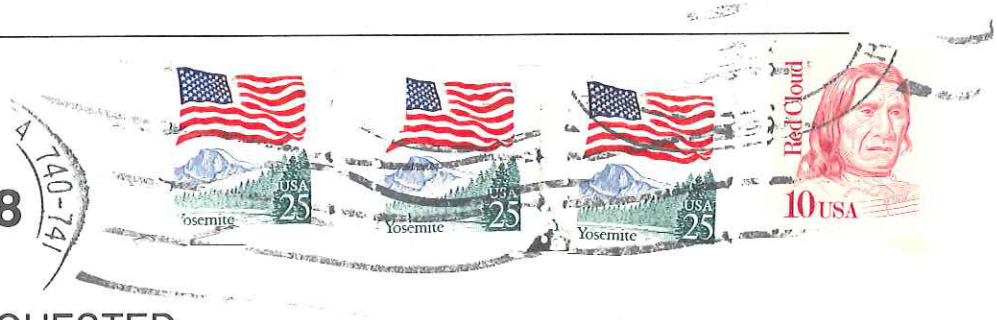
$15 special handling outside USA, Canada, and Mexico.

Write to the La Jolla address to obtain a membership form.

NewsLetter Publication Dates

Issue	Due Date For All Newsletter Material
Jul / Aug 1990	July 6, 1990
Sep / Oct 1990	September 1, 1990
Nov / Dec 1990	November 2, 1990
Jan / Feb 1990	January 4, 1991

USUS
P.O. BOX 1148
LA JOLLA, CA 92038



ADDRESS CORRECTION REQUESTED
FIRST CLASS MAIL

[redacted]