USUS NewsLetter

Serving the Pascal, Modula-2, and Portable Programming Community Vol. 5 No. 1 Jan - Feb 1991

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
William Smith Publisher

Using Procedure Vectors To Ease User Interface Design

By John M. Hughes - 1 January 1989

Email: jmh%moondog@datalog.com
jmh@coyote.datalog.com

Background

Developing large software systems with extensive menu-driven interfacing is often an arduous task at best. The common approach of using layer after layer of CASE structures or switch constructs leads to programs that are difficult to maintain, and tedious to design and debug. The programming language Modula-2 offers a simple an elegant way to completely sidestep this aspect of user interface design. The result is a simple one or two procedure deep control structure that is easy to modify and simple to implement. This is accomplished by utilizing the PROC type of Modula-2 and creating vector tables for each menu structure in the system.

Some Thoughts On Vector Tables

One of the more interesting features of Modula-2 is its ability to declare procedures as types. This, in effect, allows a programmer to write code that behaves much like the indirect operations available with certain processor instruction sets.

Vector-tables (also referred to as jump-tables) are well known to those programmers whom spend large amounts of time dealing with assembly language code. The common approach is to build a list of target addresses for various subroutines, and then jump "through" the list based on a relative offset value. This offset value is itself nothing more than a pointer into the list, which contains the addresses of subroutines to be executed with an indirect JSR (jump to subroutine) or CALL instruction. Processors with indirect forms of subroutine call instructions lend themselves to this technique directly. In assembly language programming this constitutes a powerful technique for rapidly and effi-

ciently altering program flow.

Some high-level languages, such as Modula-2 and C, also have the ability to utilize indirect calls. This article illustrates the use of the standard types PROCEDURE and PROC in Modula-2. The examples contained in this article were compiled and testing using the Fitted Software Tools Version 2.0 Modula-2 compiler system, but I have tried to make the code as generic as possible. They should compile and run with other compiler types with no modifications, since PROC is part of the original language definition according to Niklaus Wirth.

Dynamic Linked Lists Versus CASE Structures

Before we get started, perhaps a comment about linked lists is in order. Yes, there is a way to perform similar functions in Modula-2 by using dynamic linked lists, each node of which contains a pointer to particular procedure. I have chosen not to delve into this area, however, because it is already well covered in most standard texts on the language. Niklaus Wirth gives an excellent example of this technique in his book, Programming in Modula-2, and I would refer the reader there for more information. Although much has been written on the use of the Modula-2 procedure type in linked list data structures, references to its application in building vector-table type constructs appear to be rather scarce.

My main focus here is the design and construction of application dependent code based on the use of the CASE structure. In this article we will examine this alternative usage of the procedure type in conjunction with the creation of a complex user interface based on layered menus. This type of application often relies heavily on multiple CASE structures to evaluate user selections. The use of the procedure types allows the programmer to eliminate redundant CASE structures in the code and replace them with a single procedure vector dispatch handler.

The Procedure Type In Modula-2

In Modula-2, the procedure type may be declared in one of two ways: (1) as a parameter-defined type, or (2) as a parameterless type. The term "parameter-defined" is used to mean those occurrences of the procedure type where one specifically defines what parameter types will be used. An example of this would be:

```
VAR
SomeProcedure : PROCEDURE(CARDINAL,CARDINAL);
```

Then, later in the code,

```
SomeProcedure := WriteCard;
```

may be used to assign WriteCard from the standard library to the name SomeProcedure. By the same token, any procedure that accepts two parameters of type CARDINAL may be assigned to SomeProcedure.

The second form is somewhat more generic, and is coded as:

```
VAR SomeProcedure : PROC;
```

When this is used one does not need to declare parameters, but the parameters of the procedure pointed to by the variable designated as being of type PROC must match the actual parameters passed to it.

Using The PROC Type - Examples

Consider a typical problem: You are writing a rather large applications package that makes extensive use of menu screens and associated CASE structures to route the user to various portions of the code. If you took a standard approach, you would have to duplicate the code for the menu display and CASE routing of the user's selection for each and every menu in the system. For a program of even modest size this may begin to approach twenty or so menus, each with its own section of CASE code. That's a lot of wasted code, especially for those paths in the menu tree that are seldom traversed.

It would seem to make more sense to define a list of possible target procedures, and then hand it to one procedure that does nothing but select procedure calls based on input from the user. This, then, would reduce the problem to simply constructing vector-tables for each of the possible menu displays. The CASE vector structure would never change, and could be reused any number of times.

The key to this approach is to use the type PROC to build a one- dimensional array. The program show in Listing 1 illustrates the use of an array of type PROC. Notice that the procedures called have no parameters.

An important point to notice here is the way that the execution loop is tested for a valid end condition. Instead of attempting to determine which procedure is currently executing, the relative pointer value is used. This is

because there exists no type coercion mechanism between an array of type CHAR (a string) and the type PROC.

While the previous example may be interesting, and in some cases useful, the next example contained in Listing 2 shows where this technique really shines. Here we have basically the same program, only now a crude user interface has been added. When the user selects a number from the menu, the appropriate procedure is called from the vector-table. Also notice that the execution loop is now itself a separate procedure. The main body of the program simply passes an array of procedures to use as the vector-table and the relative vector pointer into the array.

One might ask at this point: "Why have a separate procedure to handle the vector dispatch?". There is a good reason for this. Because Modula-2 allows the programmer to easily pass arrays as procedure parameters, separating the vector-table dispatcher from the rest of the code allows one to give it any pre-defined array of procedure pointers. In other words, it becomes a generic, and hence reusable, procedure.

Handling Variable Size Vector Selection Lists

The reader may notice that this implementation of the technique does have a fundamental limitation: The case structure in the vector dispatch handler should have less or the same amount of choices as the vector-table array has declared elements. This lack of generalization also precludes the technique from implementation as part of a library module, unless one is willing to define very large vector-table arrays to handle most conceivable cases.

One way to resolve the array indices problem mentioned above would be to define both the vector-table array and the vector handler CASE structure for the maximum possible number of choices in a particular program application. If one inspects the declaration portion of both example programs it will be seen that this has, in fact, been done for the vector-table array. In Listing 2, also notice that the procedure Jump has the ability to utilize the entire vector-table array, but that the table itself only contains five valid vectors.

One method for trapping an "early-end" condition is illustrated in the way that menu item 6 is trapped by an IF-THEN-ELSE structure in the main body of Listing 2. The vector selection list for any given menu is simply overlaid on the vector-table array, and the trap point set for the end of the list + 1.

Towards Eliminating Redundant Code

Finally, there is one further trick in this rather interesting bag. The main body of Listing 2 could itself be made into a sub-procedure that accepts the name of a menu display array (ARRAY [0..n] OF StringType), and a vector list. This will eliminate most of the redundant menu handling code, since the programmer would now need only to define the following items for each menu in the system:

1 - The Menu Display
2 - The Vector Selection List
3 - The Menu Item Cut-Off Point (list item + 1)

The header line for this procedure might look something like this:

```
DoMenu ( Menu    : MenuDisplay;
         Vlist   : ProcList);
```

Alternatively, one could define a record structure with all the necessary elements for each menu:

```
TYPE VDef = RECORD
              MenuLine : ARRAY [0..80] OF CHAR;
              PVector  : PROC;
            END;

VAR MenuDef : ARRAY [0..n] OF VDef;
```

where n is the number of selection items in the menu.

Acknowledgements

I would like to express my appreciation to Roger Carvalho, the author of the FST compiler, for the time he spent reviewing this article and making helpful comments and corrections.

Product Reference

Fitted Software Tools
P.O. Box 867403
Plano, Texas 75086

FST Modula-2 Compiler System, Version 2.0

Listing 1 - Demonstration of type PROC as an array

```
MODULE JmpTest1;

FROM InOut IMPORT WriteString,WriteLn;

TYPE
ProcList = ARRAY [1..9] OF PROC;

VAR
VectorTo : ProcList; (* Procedure vector table  *)
ProcNum  : CARDINAL; (* Relative vector pointer *)



(* ************ Test Procedures ************************** *)

PROCEDURE Proc1;
BEGIN
WriteString("Number 1 worked....");WriteLn;
END Proc1;

PROCEDURE Proc2;
BEGIN
WriteString("Number 2 worked....");WriteLn;
END Proc2;

PROCEDURE Proc3;
BEGIN
WriteString("Number 3 worked....");WriteLn;
END Proc3;

PROCEDURE Proc4;
BEGIN
WriteString("Number 4 worked....");WriteLn;
END Proc4;

PROCEDURE Proc5;
BEGIN
WriteString("Number 5 worked....");WriteLn;
END Proc5;
```

```
(* ************************ Main Body ************************** *)

BEGIN

VectorTo[1] := Proc1;          (* Load the call table   *)
VectorTo[2] := Proc2;
VectorTo[3] := Proc3;
VectorTo[4] := Proc4;
VectorTo[5] := Proc5;
ProcNum := 1;                (* init the table pointer *)

(* call each proc in table until the end is reached *)

WHILE ProcNum < 6 DO
VectorTo[ProcNum];
INC(ProcNum);
END;

WriteLn;

END JmpTest1.

(* ************************************************************** *)
```

Listing 2 - User Interface Example

```
MODULE JmpTest2;

FROM InOut IMPORT WriteString,WriteLn,ReadCard;

TYPE
ProcList = ARRAY [1..9] OF PROC;

VAR
CallList : ProcList;
ProcNum  : CARDINAL;
ExitLoop : BOOLEAN;
UserIn   : CARDINAL;
```

```
(* ************ Test Procedures ************************** *)

PROCEDURE Number1;
BEGIN
WriteLn;
WriteString("Number 1 worked....");WriteLn;
END Number1;

PROCEDURE Number2;
BEGIN
WriteLn;
WriteString("Number 2 worked....");WriteLn;
END Number2;

PROCEDURE Number3;
BEGIN
WriteLn;
WriteString("Number 3 worked....");WriteLn;
END Number3;

PROCEDURE Number4;
BEGIN
WriteLn;
WriteString("Number 4 worked....");WriteLn;
END Number4;

PROCEDURE Number5;
BEGIN
WriteLn;
WriteString("Number 5 worked....");WriteLn;
END Number5;


(* ******************* Vector-Table Handler ********************** *)

PROCEDURE Jump (    JMPptr    : CARDINAL;
JMPList  : ProcList);

BEGIN
CASE JMPptr OF
1  :  JMPList[1];   |
2  :  JMPList[2];   |
3  :  JMPList[3];   |
4  :  JMPList[4];   |
5  :  JMPList[5];   |
6  :  JMPList[6];   |
7  :  JMPList[7];   |
8  :  JMPList[8];   |
9  :  JMPList[9];
ELSE
WriteString("Invalid Vector Pointer");WriteLn;
END;
END Jump;


(* ************************ Main Body ************************** *)

BEGIN
CallList[1] := Number1;                (* Load the call table    *)
CallList[2] := Number2;
CallList[3] := Number3;
CallList[4] := Number4;
CallList[5] := Number5;

ExitLoop := FALSE;                     (* loop control switch    *)

(* This loop will repeat the menu display until the user signals
   that an exit is desired. *)
WHILE NOT ExitLoop DO
WriteLn; WriteLn;
WriteString(" 1 - Menu Item Number One");WriteLn;
WriteString(" 2 - Menu Item Number Two");WriteLn;
WriteString(" 3 - Menu Item Number Three");WriteLn;
WriteString(" 4 - Menu Item Number Four");WriteLn;
WriteString(" 5 - Menu Item Number Five");WriteLn;
WriteLn;
WriteString(" 6 - Exit");WriteLn;
WriteLn;
WriteString(" Your Selection? -> ");
ReadCard(UserIn);
WriteLn;
IF UserIn < 6 THEN          (* test for menu exit       *)
Jump(UserIn,CallList)
ELSE
ExitLoop := TRUE
END; (* IF *)
END; (* WHILE *)
END JmpTest2.
```

The Macintosh Pascal RunTime Debugger

by
David T. Craig
736 Edgewater, Wichita, Kansas 67230

(1986)

INTRODUCTION

The Pascal RunTime Debugger, RTDebugger, is a tool that allows programmers for the Apple Macintosh computer to debug their compiled Pascal programs interactively. RTDebugger is written in Lisa Pascal.

USING THE DEBUGGER

To use RTDebugger requires a Macintosh running the program to be debugged and an external terminal which displays RTDebugger's output. RTDebugger uses the Macintosh modem port to send debugging information to the programmer. The communication parameters are: baud rate = 2400, data bits = 8, stop bits = 2, and duplex = full. I use a Macintosh XL to run the application that needs debugging and a Macintosh 512 with the FreeTerm terminal program as the external terminal.

DEBUGGER COMMANDS

The debugger supports two kinds of commands: application commands and external commands. The application commands consist of calls to RTDebugger's routine named DDT. An application debugger command is contained within conditional compile comment commands so that compiling the non-debugged application version involves only changing one flag. An example of these debugging commands is

```
{$IFC ctvDebug} DDT(dcBP,'do_AppleMenu [200]'); {$ENDC}
```

The debugger call consists of a command (in this case dcBP) and a string (in this case 'do_AppleMenu [200]') which provides specific information to the debugger. RTDebugger supports the following application code commands:

```
dcInitialize     - Initialize the debugger
dcTerminate      - Terminate the debugger
dcBP             - Tell the debugger that a routine is starting
dcEP             - Tell the debugger that a routine is ending
dcWriteMsg       - Output a message to the external terminal
dcExternalCmd    - Allow interactive commands on external terminal
```

The program Edit, whose source code is shown in another section in this document, shows how to use these commands.

External commands are commands that the programmer issues to the debugger thru the external terminal. These commands allow the programmer to watch information about executing routines, set break points on specific routines, view routine access frequencies & routine duration times, and many other items of interest to your typical Macintosh programmer.

ROUTINE NAMES AND LEVEL NUMBERS

Every routine in the application should have a debugger entry call command and an exit call command. These debugger calls tell the debugger that a routine is currently being executed by the Macintosh and allows the debugger to keep several pieces of statistical information about the routine. Each of these entry and exit calls has a message which tells the name of the routine and its level number. The routine name is simply the name of the routine as defined in the Pascal source

code. The level number assigns a unique value to the routine which the debugger uses when the programmer interactively accesses the debugged program. An example of a routine with these commands is (the debugger code is underlined)

```
PROCEDURE doInMenuBar;

  BEGIN { doInMenuBar }

    {$IFC ctvDebug} DDT(dcBP,'doInMenuBar [999]'); {$ENDC}

    doCommand(MenuSelect(myEvent.where));

    {$IFC ctvDebug} DDT(dcEP,'doInMenuBar [999]'); {$ENDC}

  END;  { doInMenuBar }
```

EXTERNAL DEBUGGER COMMANDS

Access to the external debugger is thru the dcExternalCmd command to the debugger routine DDT. The sample program Edit tests for the Shift, Option and Command keys in the main event loop and if these keys are down the dcExternalCmd command is sent to the debugger. The code for this test is in the Edit routine handleDebuggerActivation.

When the debugger wants an external command it prompts for a line of text from the programmer on the external terminal. The text line is then processed and the appropriate debugger external command is performed. These commands are as follows

Command	Parameter(s)	Description
HELP/?	command	Display help information
GO		Continue the application
W	min_lev max_lev [P]	Set watch/trace level
GW		Get watch/trace level
PS	[L/F/D]	Show routine statistics
SBP	proc_num B/E	Set breakpoint
CBP	[proc_num]	Clear breakpoints
DBP		Display all breakpoints
DD		Display date & time
MI		Display machine info
WIND		Show window information
VOL		Show volume information
CAT	[volume_name]	Show volume catalog
CV	number	Convert number to hex/dec
DM	address [count]	Display machine memory
DFM		Display free memory
DTM		Display top of memory
CM		Compact memory
KPM		Kill purgable memory
IL	address [count]	Instruction list
HD	[res_mask]	Dump application Heap
WH	trap_name	Show trap address
DR		Display CPU registers
RB		Reboot machine
ES		Exit to machine shell
GOMB		Exit to MacsBug debugger
DV		Show debugger version

(Parameters enclosed in [] are optional, others are required)

RTDebugger is currently not finished, but all of the important commands are implemented. The others state that they are not finished when the programmer tries to invoke them.

SAMPLE DEBUGGER SESSION

This sample debugger session was performed on the Edit program which is listed below.

```
*************************************************
***                                           ***
***                 Welcome to the            ***
***           Pascal RunTime Debugger         ***
***                                           ***
***                by David T Craig           ***
***                                           ***
*************************************************

Application...: TinyEdit
Date.........: 30-Jul-1986 16:24

Initializing the application stuff
  Creating the edit window

USER KEY BREAK in handleDebuggerActivation [Hi]

> ?

Command  Parameter(s)          Description
-------  --------------------  ------------------------------
HELP/?   command               Display help information
GO                             Continue the application
W        min_lev max_lev [P]   Set watch/trace level
GW                             Get watch/trace level
PS       [L/F/D]               Show routine statistics
SBP      proc_num B/E          Set breakpoint
CBP      [proc_num]            Clear breakpoints
DBP                            Display all breakpoints
DD                             Display date & time
MI                             Display machine info
WIND                           Show window information
VOL                            Show volume information
CAT      [volume_name]         Show volume catalog
CV       number                Convert number to hex/dec
DM       address [count]       Display machine memory
DFM                            Display free memory
DTM                            Display top of memory
CM                             Compact memory
KPM                            Kill purgable memory
IL       address [count]       Instruction list
HD       [res_mask]            Dump application Heap
WH       trap_name             Show trap address
DR                             Display CPU registers
RB                             Reboot machine
ES                             Exit to machine shell
GOMB                           Exit to MacsBug debugger
DV                             Show debugger version

(Parameters enclosed in [] are optional - others are required)

> DV
MacRTDebugger by David Craig Ver. 0.50 [30-Jul-86 16:01:25]
> ? CV
CV       number                Convert number to hex/dec
> CV 200
200 = $000000C8
```

```
> CV $C8
200 = $000000C8
> GO

Leaving the RunTime Debugger...

USER KEY BREAK in handleDebuggerActivation [Hi]

> GW
Minimum watch level = 9990
Maximum watch level = 9999
> W 1010 1010 P
> GW
Minimum watch level = 1010
Maximum watch level = 1010
Watching proc entry and exit points
> GO

Leaving the RunTime Debugger...

---> xKeyDownEvent
Performing a key down event
theKey = N
<--- xKeyDownEvent
---> xKeyDownEvent
Performing a key down event
theKey = o
<--- xKeyDownEvent
---> xKeyDownEvent
Performing a key down event
theKey = w
<--- xKeyDownEvent

USER KEY BREAK in handleDebuggerActivation [Hi]

> W 1000 1040
> GW
Minimum watch level = 1000
Maximum watch level = 1040
> PS
Number of Symbol Table procs = 9
Sorting by name..

 #  ProcName                        Level Freq Time       T% Attr
--- ------------------------------- ----- ---- --------- -- ----
  1 doCommand.....................    500    1        57  3
  2 doInMenuBar...................    999    1       134  7
  3 doInSysWindow.................    999    1        46  2
  4 do_AppleMenu..................    200    1        55  2
  5 init_ApplicationStuff.........   9999    1        68  3
  6 xActivateEvent................   1030    3         4  0
  7 xKeyDownEvent.................   1010   94      1313 70
  8 xMouseDownEvent...............   1000    2       184  9
  9 xUpdateEvent..................   1040    2         4  0
> ? SBP
SBP      proc_num B/E         Set breakpoint
> SBP 9 B
> GBP
What ?
> DBP
```

```
 #  ProcName                          Level B E
--- -------------------------------- ----- - -
  1 xUpdateEvent                      1040 B
> GO

Leaving the RunTime Debugger...

Performing a mouse down event
Performing an activate event
"TinyEdit Sample Window" is becoming inactive
Performing a mouse down event
Performing an activate event
"TinyEdit Sample Window" is becoming active

USER BREAK POINT in xUpdateEvent [BEGIN]

> PS
Number of Symbol Table procs = 9
Sorting by name
 #  ProcName                          Level Freq Time      T% Attr
--- -------------------------------- ----- ---- --------- -- ----
  1 doCommand.....................    500    2        99  4
  2 doInMenuBar...................    999    2       255 10
  3 doInSysWindow.................    999    2        84  3
  4 do_AppleMenu..................    200    2        96  4
  5 init_ApplicationStuff.........   9999    1        68  2
  6 xActivateEvent................   1030    5        48  2
  7 xKeyDownEvent.................   1010   94      1313 56
  8 xMouseDownEvent...............   1000    4       362 15
  9 xUpdateEvent..................   1040    2         4  0 B
> MI
Machine type.........: Lisa
Machine RAM (K)......: 908
ROM Version..........: 130
ROM Type.............: 64K
CPU Type.............: 68000
File System type.....: MFS
System Version.......: -1
Keyboard Version.....: 0
Switcher status......: Inactive
Servant status.......: Inactive
Journal status.......: Inactive
Monkey status........: Inactive
Stack Sniffer status.: Active
> DFM
Free memory (bytes) = 812576
> VOL
 #  VolumeName                 Creation-Date     Last-Mod-Date     Files Attr
--- -------------------------- ----------------- ----------------- ----- ----
  1 ProFile                    29-Jul-1986 15:24 01-Jan-1980 13:17     8
  2 TESTER                     24-Jul-1986 14:13 30-Jul-1986 16:23     3
> GO

Leaving the RunTime Debugger...

USER KEY BREAK in handleDebuggerActivation [Hi]
```

```
> W 0 9999
> GO

Leaving the RunTime Debugger...

Performing a mouse down event
Mouse down in Menu Bar area
User issued a menu command
theMenu = 0
theItem = 0
Performing a mouse down event
Mouse down in application window Content area
Mouse down in application window Grow area
Performing a mouse down event
Mouse down in Menu Bar area
User issued a menu command
theMenu = 257 (Edit)
theItem = 2
Edit menu item selected
Edit action = Copy
Performing a mouse down event
Mouse down in Menu Bar area
User issued a menu command
theMenu = 256 (File)
theItem = 1
File menu item selected
Terminating the application

The Debugger says "Have a nice day"
```

EDIT SOURCE CODE LISTING

```
PROGRAM Edit;

  {------------------------------------------------------------------------}
  {                            --- Edit ---                                }
  {           A small sample application written in Pascal                 }
  {                    by Macintosh User Education                         }
  {                         (12 December 83)                               }
  {------------------------------------------------------------------------}

  {------------------------------------------------------------------------}
  {              Modified by David Craig (21 July 1986)                    }
  {------------------------------------------------------------------------}

   {$M+ } { Turn ON  Macintosh code generation }
   {$X- } { Turn OFF RunTime stack expansion (Lisa concept) }
   {$U- } { Turn OFF Lisa Libraries }

   USES   {$L-} { Turn OFF unit output listing }
          {$U MAC/Obj/MemTypes     } MemTypes,        { Memory Data Types }
          {$U MAC/Obj/QuickDraw    } QuickDraw,       { QuickDraw         }
          {$U MAC/Obj/OSIntf       } OSIntf,          { Operating System  }
          {$U MAC/Obj/ToolIntf     } ToolIntf,        { ToolBox           }
          {$U MAC/Obj/PackIntf     } PackIntf,        { Package Manager   }

          {$U MAC/P3/UMacHardware  } P3_MacHardware,  { Hardware Manager  }
          {$U MAC/P3/UStringUtils  } P3_StringUtils,  { String Utilities  }
          {$U MAC/P3/UNumberUtils  } P3_NumberUtils,  { Number Utilities  }
          {$U MAC/P3/URTDebugger   } P3_RTDebugger;   { RunTime Debugger  }
          {$L+} { Turn ON unit output listing }
```

```
CONST  lastMenu  =   3; { number of menus }
       appleMenu =   1; { menu ID for desk accessory menu }
       fileMenu  = 256; { menu ID for File menu }
       editMenu  = 257; { menu ID for Edit menu }

VAR    userIsDone  : BOOLEAN;
       myMenus     : ARRAY [1..lastMenu] OF MenuHandle;
       screen_Rect : Rect;
       dragRect    : Rect;
       pRect       : Rect;
       wRecord     : WindowRecord;
       myWindow    : WindowPtr;
       hTE         : TEHandle;

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

PROCEDURE doCommand(mResult: LongInt);
   VAR theMenu : INTEGER;  theItem : INTEGER;

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

   PROCEDURE do_AppleMenu;
     VAR name : STR255;  refNum : INTEGER;
     BEGIN { do_AppleMenu }
       {$IFC ctvDebug}
       DDT(dcBP,'do_AppleMenu [200]');
       DDT(dcWriteMsg,'Apple menu item (desk accessory) selected [200]');
       {$ENDC}

       GetItem(myMenus[1],theItem,name);
       refNum := OpenDeskAcc(name);

       {$IFC ctvDebug}
       DDT(dcWriteMsg,CONCAT('Desk accessory name = "',name,'" [200]'));
       DDT(dcEP,'do_AppleMenu [200]');
       {$ENDC}
     END;  { do_AppleMenu }

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

   PROCEDURE do_FileMenu;
     BEGIN { do_FileMenu }
       {$IFC ctvDebug}
       DDT(dcBP,'do_FileMenu [300]');
       DDT(dcWriteMsg,'File menu item selected [300]');
       {$ENDC}

       userIsDone := TRUE; { Quit }

       {$IFC ctvDebug} DDT(dcEP,'do_FileMenu [300]'); {$ENDC}
     END;  { do_FileMenu }

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

   PROCEDURE do_EditMenu;
     BEGIN { do_EditMenu }
       {$IFC ctvDebug}
       DDT(dcBP,'do_EditMenu [400]');
       DDT(dcWriteMsg,'Edit menu item selected [400]');
       {$ENDC}

       IF NOT(SystemEdit(theItem-1))
         THEN
```

```
            BEGIN
              SetPort(myWindow);

              {$IFC ctvDebug}
              CASE theItem OF
                1: ddtString := 'Cut';
                2: ddtString := 'Copy';
                3: ddtString := 'Paste';
              END;
              DDT(dcWriteMsg,CONCAT('Edit action = ',ddtString,' [400]'));
              {$ENDC}

              CASE theItem OF
                1: TECut  (hTE);
                2: TECopy (hTE);
                3: TEPaste(hTE);
              END;
            END;

        {$IFC ctvDebug} DDT(dcEP,'do_EditMenu [400]'); {$ENDC}
      END;  { do_EditMenu }

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

    BEGIN { doCommand }
      {$IFC ctvDebug}
      DDT(dcBP,'doCommand [500]');
      DDT(dcWriteMsg,'User issued a menu command [500]');
      {$ENDC}

      theMenu := HiWord(mResult);
      theItem := LoWord(mResult);

      {$IFC ctvDebug}
      NumToString(theMenu,ddtString);
      CASE theMenu OF
        appleMenu : ddtString := CONCAT(ddtString,' (Apple)');
        fileMenu  : ddtString := CONCAT(ddtString,' (File)');
        editMenu  : ddtString := CONCAT(ddtString,' (Edit)');
      END;
      DDT(dcWriteMsg,CONCAT('theMenu = ',ddtString,' [500]'));
      NumToString(theItem,ddtString);
      DDT(dcWriteMsg,CONCAT('theItem = ',ddtString,' [500]'));
      {$ENDC}

      CASE theMenu OF
        appleMenu : do_AppleMenu;
        fileMenu  : do_FileMenu;
        editMenu  : do_EditMenu;
      END; { of menu case }

      HiliteMenu(0);

      {$IFC ctvDebug} DDT(dcEP,'doCommand [500]'); {$ENDC}
    END; { of doCommand }

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

 PROCEDURE initialize;

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

   PROCEDURE init_SystemStuff;
```

```
{-------------------------------------------------------------}
 PROCEDURE setUpMenus; { Once-only initialization for menus }
    VAR i : INTEGER;  appleTitle : STRING[1];
    BEGIN
      appleTitle    := '?';
      appleTitle[1] := CHR(AppleMark);

      myMenus[1]    := NewMenu(appleMenu,appleTitle);
      AddResMenu(myMenus[1],'DRVR'); { desk accessories }

      myMenus[2]    := GetMenu(fileMenu);
      myMenus[3]    := GetMenu(editMenu);

      FOR i := 1 TO lastMenu DO
        InsertMenu(myMenus[i],0);

      DrawMenuBar;
    END; { of setUpMenus }
{-------------------------------------------------------------}
 BEGIN { init_SystemStuff }
   { Initialize the low-level memory items first }

   MaxApplZone;   { Expand application heap zone to the max. }

   MoreMasters;   { Allocate several 'master pointer blocks' }
   MoreMasters;   {  so that they occupy low memory in order }
   MoreMasters;   {  to avoid memory fragmentation later on. }
   MoreMasters;  MoreMasters;  MoreMasters;

   { Initialize the many Macintosh Managers }

   InitGraf(@ThePort);  { Init Bill's QuickDraw            }
   InitFonts;           { Init the Font Manager            }
   InitWindows;         { Init the Window & Event Manager  }
   InitMenus;           { Init the Menu Manager            }
   TEInit;              { Init the Text Editor             }
   InitDialogs(NIL);    { Init the Dialog Manager          }
   InitCursor;          { Init the cursor stuff            }

   { Setup the application menus }
   setUpMenus;

   { Empty the Macintosh Event queue }
   FlushEvents(EveryEvent,0);
 END;  { init_SystemStuff }
{-------------------------------------------------------------}
 PROCEDURE init_ApplicationStuff;
   VAR wBounds : Rect;  wTitle : Str255;
   BEGIN { init_ApplicationStuff }
     { Initialize the debugger }

     {$IFC ctvDebug}
     DDT(dcInitialize,'TinyEdit');
     DDT(dcWriteMsg,'Initializing the application stuff [9999]');
     DDT(dcBP,'init_ApplicationStuff [9999]');
     {$ENDC}

     userIsDone := FALSE;
     screen_Rect := screenBits.bounds;
     SetRect(dragRect,4,24,screen_Rect.right-4,screen_Rect.bottom-4);
```

```
      {$IFC ctvDebug}
      DDT(dcWriteMsg,'  Creating the edit window [9999]');
      {$ENDC}

      GetIndString(wTitle,128,1); { 'TinyEdit Sample Window' }

      WITH wBounds DO
        BEGIN
          Left   := 20;
          Top    := 50;
          Right  := screen_Rect.Right  - 20;
          Bottom := screen_Rect.Bottom - 20;
        END;

      myWindow := NewWindow(@wRecord,wBounds,wTitle,
                            TRUE,RDocProc,POINTER(-1),FALSE,0);

      SetPort(myWindow);

      pRect := thePort^.portRect;
      InsetRect(pRect,4,0);
      hTE := TENew(pRect,pRect);

      {$IFC ctvDebug} DDT(dcEP,'init_ApplicationStuff [9999]'); {$ENDC}
    END;  { init_ApplicationStuff }

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

  BEGIN { initialize }
    init_SystemStuff;
    init_ApplicationStuff;
  END;  { initialize }

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

PROCEDURE terminate;
  BEGIN { terminate }
    {$IFC ctvDebug}
    DDT(dcBP,'terminate [9999]');
    DDT(dcWriteMsg,'Terminating the application [9999]');
    {$ENDC}

    { ??? Terminate application stuff here ??? }

    {$IFC ctvDebug}
    DDT(dcEP,'terminate [9999]');
    DDT(dcTerminate,'');
    {$ENDC}
  END;  { terminate }

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

PROCEDURE handleUserEvents;
  VAR eventIsGood : BOOLEAN;  myEvent : EventRecord;

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

  PROCEDURE xMouseDownEvent;
    VAR code : INTEGER;  whichWindow : WindowPtr;

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

    PROCEDURE doInMenuBar;
      BEGIN { doInMenuBar }
        {$IFC ctvDebug}
```

```
      DDT(dcBP,'doInMenuBar [999]');
      DDT(dcWriteMsg,'Mouse down in Menu Bar area [999]');
      {$ENDC}

      doCommand(MenuSelect(myEvent.where));

      {$IFC ctvDebug} DDT(dcEP,'doInMenuBar [999]'); {$ENDC}
    END;  { doInMenuBar }

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

 PROCEDURE doInSysWindow;
   BEGIN { doInSysWindow }
     {$IFC ctvDebug}
     DDT(dcBP,'doInSysWindow [999]');
     DDT(dcWriteMsg,'Mouse down in System window area [999]');
     {$ENDC}

     SystemClick(myEvent,whichWindow);

     {$IFC ctvDebug} DDT(dcEP,'doInSysWindow [999]'); {$ENDC}
   END;  { doInSysWindow }

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

 PROCEDURE doInDrag;
   BEGIN { doInDrag }
     {$IFC ctvDebug}
     DDT(dcBP,'doInDrag [999]');
     DDT(dcWriteMsg,'Mouse down in application window Drag area [999]');
     {$ENDC}

     DragWindow(whichWindow,myEvent.where,dragRect);

     {$IFC ctvDebug} DDT(dcEP,'doInDrag [999]'); {$ENDC}
   END;  { doInDrag }

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

 PROCEDURE doInGrow;
   BEGIN { doInGrow }
     {$IFC ctvDebug}
     DDT(dcBP,'doInGrow [999]');
     DDT(dcWriteMsg,'Mouse down in application window Grow area [999]');
     {$ENDC}

     IF whichWindow <> FrontWindow
       THEN
         SelectWindow(whichWindow)
       ELSE
         BEGIN
           GlobalToLocal(myEvent.where);
           TEClick(myEvent.where,FALSE,hTE);
         END;

     {$IFC ctvDebug} DDT(dcEP,'doInGrow [999]'); {$ENDC}
   END;  { doInGrow }

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

 PROCEDURE doInContent;
   BEGIN { doInContent }
     {$IFC ctvDebug}
     DDT(dcBP,'doInContent [999]');
```

```
        DDT(dcWriteMsg,
            'Mouse down in application window Content area [999]');
        {$ENDC}

        doInGrow;

        {$IFC ctvDebug} DDT(dcEP,'doInContent [999]'); {$ENDC}
      END;  { doInContent }

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

    BEGIN { xMouseDownEvent }
      {$IFC ctvDebug}
      DDT(dcBP,'xMouseDownEvent [1000]');
      DDT(dcWriteMsg,'Performing a mouse down event [1000]');
      {$ENDC}

      code := FindWindow(myEvent.where,whichWindow);

      CASE code OF
        inMenuBar    : doInMenuBar;
        inSysWindow  : doInSysWindow;
        inDrag       : doInDrag;
        inGrow       : doInGrow;
        inContent    : doInContent;
      END; { of code case }

      {$IFC ctvDebug} DDT(dcEP,'xMouseDownEvent [1000]'); {$ENDC}
    END;  { xMouseDownEvent }

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

  PROCEDURE xKeyDownEvent;
    VAR theKey : CHAR;
    BEGIN { xKeyDownEvent }
      {$IFC ctvDebug}
      DDT(dcBP,'xKeyDownEvent [1010]');
      DDT(dcWriteMsg,'Performing a key down event [1010]');
      {$ENDC}

      IF myWindow = FrontWindow
        THEN
          BEGIN
            theKey := CHR(myEvent.message MOD 256);

            {$IFC ctvDebug}
            IF (theKey >= ' ') AND (theKey <= '~')
              THEN
                BEGIN
                  ddtString  := '?';
                  ddtString[1] := theKey;
                END
              ELSE
                BEGIN
                  NumToString(ORD(theKey),ddtString);
                  ddtString := CONCAT('<',ddtString,'>');
                END;
            DDT(dcWriteMsg,CONCAT('theKey = ',ddtString,' [1010]'));
            {$ENDC}

            TEKey(theKey,hTE);
          END;
```

```
    {$IFC ctvDebug} DDT(dcEP,'xKeyDownEvent [1010]'); {$ENDC}
  END;  { xKeyDownEvent }

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

PROCEDURE xAutoKeyEvent;
  BEGIN { xAutoKeyEvent }
    {$IFC ctvDebug} DDT(dcBP,'xAutoKeyEvent [1020]'); {$ENDC}

    xKeyDownEvent;

    {$IFC ctvDebug} DDT(dcEP,'xAutoKeyEvent [1020]'); {$ENDC}
  END;  { xAutoKeyEvent }

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

PROCEDURE xActivateEvent;
  BEGIN { xActivateEvent }
    {$IFC ctvDebug}
    DDT(dcBP,'xActivateEvent [1030]');
    DDT(dcWriteMsg,'Performing an activate event [1030]');
    ddtString := WindowPeek(myEvent.Message)^.TitleHandle^^;
    {$ENDC}

    IF ODD(myEvent.Modifiers)
      THEN
        BEGIN { Window is becoming active }
          {$IFC ctvDebug}
          DDT(dcWriteMsg,
              CONCAT('"',ddtString,'" is becoming active [1030]'));
          {$ENDC}

          TEActivate(hTE);
        END
      ELSE
        BEGIN{ Window is becoming inactive }
          {$IFC ctvDebug}
          DDT(dcWriteMsg,
              CONCAT('"',ddtString,'" is becoming inactive [1030]'));
          {$ENDC}

          TEDeactivate(hTE);
        END;

    {$IFC ctvDebug} DDT(dcEP,'xActivateEvent [1030]'); {$ENDC}
  END;  { xActivateEvent }

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

PROCEDURE xUpdateEvent;
  BEGIN { xUpdateEvent }
    {$IFC ctvDebug}
    DDT(dcBP,'xUpdateEvent [1040]');
    DDT(dcWriteMsg,'Performing an update event [1040]');
    {$ENDC}

    SetPort(myWindow);

    {$IFC ctvDebug}
    ddtString := WindowPeek(myEvent.Message)^.TitleHandle^^;
    DDT(dcWriteMsg,CONCAT('Updating window "',ddtString,'" [1040]'));
    {$ENDC}
```

```
      BeginUpdate(myWindow);
      TEUpdate(thePort^.portRect,hTE);
      EndUpdate(myWindow);

      {$IFC ctvDebug} DDT(dcEP,'xUpdateEvent [1040]'); {$ENDC}
    END;  { xUpdateEvent }

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

 {$IFC ctvDebug}
 PROCEDURE handleDebuggerActivation;
   CONST { Macintosh keyboard special key codes }
         kcCommandKey = 55; { Command }
         kcShiftKey   = 56; { Shift   }
         kcOptionKey  = 58; { Option  }
   VAR   keyboard : KeyMap; { Macintosh keyboard key state map }
   BEGIN { handleDebuggerActivation }
     GetKeys(keyboard); { Get the speical key states }

     IF keyboard[kcCommandKey] AND
        keyboard[kcShiftKey ] AND
        keyboard[kcOptionKey ]
       THEN { Enter the Debugger if the activation keys are pressed }
         DDT(dcExternalCmd,'handleDebuggerActivation <Hi> [0]'); { DDT.. }
   END;  { handleDebuggerActivation }
{$ENDC}

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

 BEGIN { handleUserEvents }
   REPEAT { Wait for a user event }
     BEGIN
       SystemTask;  { Let the Mac do its own thing for a little while }
       TEIdle(hTE); { Same for the Text Edit manager }

       {$IFC ctvDebug} handleDebuggerActivation; {$ENDC}

       eventIsGood := GetNextEvent(everyEvent,myEvent); { Fetch an event }
     END;
   UNTIL eventIsGood;

   CASE myEvent.What OF { Handle the user event }
     mouseDown   : xMouseDownEvent;
     keyDown     : xKeyDownEvent;
     autoKey     : xAutoKeyEvent;
     activateEvt : xActivateEvent;
     updateEvt   : xUpdateEvent;
   END; { of event case }
 END;  { handleUserEvents }

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

BEGIN { main program }
  initialize;           { Initialize the application }

  REPEAT                { Handle user events until the user is done }
    handleUserEvents;
  UNTIL userIsDone;

  terminate;            { Terminate the application }
END.

{                         That's all, Folks...                    }
```

Et tu, Absolut?

David T. Craig
736 Edgewater, Wichita, Kansas 67230
21 August 1990

All programmers expect the absolute value function in Pascal, ABS, to return a positive value (zero is considered positive). But this fact is not always true. The absolute value function can turn treacherous, like Caesar's friend Brutus, and return a negative value!

For all computers using two's complement arithmetic the value of ABS(- 32768) is -32768. This startling answer is a result of the non-symmetric nature of two's complement numbers about the value 0. For 16 bit integers the numeric range is -32768 to +32767. The negative side contains 32,768 distinct values while the positive side contains only 32,767 values. Since +32767 is the largest positive value the expected value for ABS(-32768) would be +32768 which is clearly not in the correct range.

The value of ABS(-32768) is negative because two's complement arithmetic wraps around for this special case. Seen in binary -32768 is 1000000000000000. The two's complement of this value is obtained by negating the value (0111111111111111) and adding 1. When 1 is added a carry ripples through the bits transforming all the 1 bits to 0 bits and finally transforming the most significant bit, a 0, to a 1. Therefore, the two's complement of 1000000000000000 is 1000000000000000, itself!

This unexpected behavior also applies to 32 bit integers and the 32 bit value -2147483648. This feature applies to any computer language that uses two's complement arithmetic and is not solely a Pascal problem. Pascal programmers may wish to special case this function by testing for the special values that generate negative ABS function results.

```
FUNCTION my_ABS (x : INTEGER) : INTEGER;

BEGIN
  IF x = -MaxInt-1 THEN
    my_ABS := ... some value ...
  ELSE
    my_ABS := ABS(x);
END;
```

To handle this numeric anomaly one could use bigger integers. For example, instead of returning a 16 bit integer, you could return a 32 bit integer which can hold the value +32768. Or you could make certain that the values you pass to the ABS function never include -32768.

Historical note: The name for this paper originated in Shakespeare's Julius Caesar (Act II, Scene I). Julius Caesar supposedly said this to Marcus Brutus, an assassin and friend, during Caesar's assassination on the Ides of March, 44 B.C. (March 15). This statement is historically incorrect. Caesar did speak to Brutus but spoke in Greek, not Latin (Suetonius, The Twelve Caesars).

That's all, folks ...

Editor's Note:

When doing arithmetic with -32768 (or the 32 bit equivalent) on machines that check for integer overflow, you will likely get an overflow on almost any operation involving -32768. As an example, one way to check for the situation described in this article is:

IF (n < 0) AND (n = -n) THEN
... have -32768 or equivalent value ...;

But the negate operation will give an overflow, which if checked by your machine, will abort the program. Thus the safer method of checking is as was described in David's example: check if n = -32768.

Board Meeting Minutes (December 5, 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 December 5, 1990.

Present at the meeting were:

User	ID Name
73447,2754	Henry Baumgarten (Henry)
71016,1203	Stephen Pickett (sfbp)
72767,622	Tom Cattrall (TomC)
73767,3521	Keith Frederick (KeithF)
76702,513	Harry Baya (Harry)
73007,173	William Smith (Wm)

Henry Baumgarten started the board meeting at 6:37 PM PST.

Topics discussed were:

I. Elections

Henry asked if there was a committee report. Stephen Pickett replied "uh oh" and poetically stated that he had not prepared the report. Henry then stated that progress must be made on the elections since they are scheduled for February and that a slate be put in the newsletter soon.

Tom Cattrall replied that there are three vacancies and currently three candidates so far: Bob Spitzer, Keith Frederick, and Felix Bearden.

Henry asked for any further nominations and stated if there weren't then asked if there would be a motion to cease nominations. Stephen indicated his utter embarrassment over forgetting to prepare the report and asked for a 48 hour extension to find additional names. Otherwise he said, the three current candidates get elected "ipso facto."

Tom replied, saying that whoever he hears from by this weekend will be put on the slate of candidates. Henry said he would accept that as a motion.

Tom Cattrall, Stephen Pickett, Harry Baya, and William Smith voted in favor and there were none opposed.

William Smith then asked who is going to count the ballots. William noted that he counted them one year because the ballots went to the La Jolla PO box but since he is not picking up the mail now the ballots should be sent to the address of whoever is picking up the mail. Henry asked for volunteers and said he would do it if no one else would. Tom Cattrall stated he could do it and Henry agreed and asked if there were any objections. There were none.

II. Submitting the bylaws

Henry asked if this would appear on the same ballot and Tom Cattrall replied that it should so that the voting procedures are gone through only once. Tom then asked if the Board should use the proposal given by Felix Bearden on MUSUS. Henry agreed on both accounts and then asked for objections. There were none.

III. Demise of JPAM (Journal of Pascal, Ada, and Modula-2)

Tom Cattrall started off by saying that it looks like the agreement with JPAM can be transferred over to CLM (Computer Language Magazine) at the same rates and terms. Tom then said if the Board wants to do that then we should move and vote to transfer to CLM as Felix proposed.

Henry noted that Felix also proposes an increase of $6 for Canadian and Mexican members and $15 for international members. William Smith commented that the increases were for international postage. Henry then said that these are the standard extra amounts that CLM tacks on to subscriptions from the various countries. Summing it up, Henry said that CLM is offering a subscription to USUS members for $20 plus any added postage.

Stephen Pickett indicated he was pleased with that saying that those are the prices that he though JPAM should have been charging all along. Henry then asked for any additional comments on the motion.

Tom Cattrall then moved to transfer the JPAM arrangement to CLM as stated by Felix Bearden.

Tom Cattrall then, for the record, posted Felix's motion:
To adopt CLM as the official Journal of USUS;
To offer CLM as a standard member benefit to all USUS members;
To increase Canadian and Mexican dues by $6 to cover postage;
To increase other international dues by $15 to cover postage;
To process all subscriptions now being held by the administrator and Treasurer to CLM.

The cost of subscriptions is $20/year (plus any extra for postage).

Henry commented that USUS may or may not get some of the other things that JPAM promised but that this (CLM) is a different journal, probably more stable and widely circulated and that the deal sounded fair to him and asked for further opinions.
William Smith mentioned that CLM deals with more than just structured languages, C and BASIC for example, and that it published 12 times year instead of 6. Also, he said that there's more in it than JPAM and the price (offered) is good.

Henry replied that he didn't care much for the issue when he first scanned it. However, after looking at other issues he has changed his mind and said he agreed with William. Henry then called for further discussion, of which there was none, then he called for a vote.

William Smith, Tom Cattrall, Stephen Pickett, and Harry Baya all voted in favor. There were no opposing votes. The motion was passed.

IV. Printing of the Newsletter

Tom Cattrall began, saying that he and William Smith compared costs and both are about the same. Tom went on to say that Felix suggested that either the editor or the administrator handle the printing. But, Tom commented, both Felix and himself are very tight for time and feel that William's offer to handle it is the best approach. Tom ended saying that the only question he and William had is how their cost quotes compare with what is currently being paid. Tom then moved to allow William Smith to handle the printing and mailing duties.

Willaim agreed with Tom's summing up of the situation.

Stephen Pickett, Harry Baya, William Smith, and Tom Cattrall all voted in favor. There were no opposing votes. The motion was passed.

V. Felix's motion 4: to pay for the CLM ads in progress

Tom Cattrall moved to have Bob Clark pay the agency for the three ads and authorize the agency to run the third advertisement in the February issue of CLM. William discussed the pricing. He indicated that the price is the same as agreed with JPAM and that CLM has a much wider distribution than JPAM. He also indicated in the third ad, if possible, to change the reference from JPAM to CLM. Stephen suggested that perhaps USUS should have more ads at that price. Henry replied though, that it is not likely to do that and there may be the cost of a new film to correct the old advertisement.

William Smith commented that, according to Felix, the correction is less than $50. Henry asked for any further discussion. Tom Cattrall said he would amend the motion to authorize the new film to change to JPAM reference to CLM.

Henry called for a vote. William Smith, Tom Cattrall, Harry Baya, and Stephen Pickett all voted in favor. There were no opposing votes. The motion passed.

The meeting adjourned at 7:51 PM PST.

NEXT MEETING

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

Minutes submitted by: Keith R. Frederick

Board Meeting Minutes (January 9, 1991)

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 January 9, 1991.

Present at the meeting were:

User	ID Name
71016,1203	Stephen Pickett (sfbp)
72767,622	Tom Cattrall (TomC)
74076,1715	Felix Bearden (felix)
73760,3521	Keith Frederick (KeithF)
73447,2754	Henry Baumgarten (Henry)
72747,3126	Bob Clark (BobC)
75226,3643	Bob Spitzer (BobS)

The Board Meeting started at 6:34 PM PST. Topics discussed were:

I. Status of the Newsletter

Tom Cattrall stated he hadn't received statements from those running in the elections until late December and should have sent the Newsletter out earlier and held the election materials until the next issue. However, Tom said, the Newsletter will be sent to William Smith next week. Time will then be needed for printing and mailing.

Felix Bearden asked who gets the mailing labels. Tom answered that William Smith does.

II. Elections

Henry Baumgarten asked what the deadline is for the return of ballots. Tom Cattrall responded that the end of February would be necessary to allow for a round trip of printing, mailing, and replies.

III. Advertisement

Henry Baumgarten asked Bob Clark about the finances of USUS. Bob answered that as of 1-3-91, with all bills paid, USUS has $5583.15. Henry asked Bob if that included the advertisement. Bob responded that it did.

Tom Cattrall then asked if there are any more advertisements scheduled and whether there have been any responses yet to the current ad. Felix Bearden replied that he had not received any mail from California lately and thus didn't know if there were any "punch card" results.

However, Felix said that he received two queries from the JPAM ad and that he (Felix) called the people who handle the JPAM ads and found that no punch cards were included in that issue. Because of this, Felix stated, they are refunding some amount of money and that since the ad agency is also printing new stationary they will be crediting the USUS account with that expense

when they bill USUS.

IV. Status of MUSUS

Tom Cattrall started off by saying that he is now a SysOp and that traffic has built up recently but nothing spectacular. Also, Tom said, that there are some new people that want to help build up the forum because they like having a forum for Modula-2.

Henry commented that he has downloaded all the messages and was impressed with the Modula-2 and Modula-3 discussions.

NEXT MEETING

The Board adjourned and agreed to meet again at 6:30 PM PST / 7:30 MST / 8:30 CST / 9:30 EST February 13, 1991 in Room 1 of the MUSUS conference facility.

Minutes submitted by: Keith R. Frederick

Treasurer's Report
by Robert E. Clark, Treasurer

January 1991

```
Bank Balance      $6,283.15 12-31-90

Income - January 1991

Dues:                          (new/renew)
  Student                0.00      0/0
  General                0.00      0/0
  Professional           0.00      0/0
  Institutional          0.00      0/0

Other Income:
  CIS                    0.00
  Library fees           0.00
                        -------

Total Income:      $     0.00
```

```
Expenses - January 1991

  Bank charges            1.82
  Newsletter            525.44
  Mail from La Jolla      0.00
  Refunds                 0.00
  Reimbursements          0.00
  JPAM/CL Adv.          700.00
                        --------

Total Expenses      $1,227.26

Bank Balance        $5,055.89 1-31-91
```

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
tomc@techbook.com

NewsLetter Publisher : William Smith

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
Mar / Apr 1991	March 1, 1991
May / Jun 1991	May 1, 1991
Jul / Aug 1991	July 1, 1991
Sep / Oct 1991	September 1, 1991
Nov / Dec 1991	November 1, 1991

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

ADDRESS CORRECTION REQUESTED
FIRST CLASS MAIL

[redacted]