USUS NewsLetter

Mar / Apr 1990 All The News that Fits, We Print Volume 4 Number 2

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, Ed.D. Publisher

Letter From David Craig

Fm: Hays Busch USUS Admin. 70260,306

In a private letter to me dated 10/31/89, David Craig makes the following comments regarding articles in the most recent NewsLetter:

- -

In regard to Alex Kleider's "The Prez Sez" article, I have several comments.

I think USUS should expand its programming horizon to include not only UCSD-Pascal but other Pascals and Pascal-derived languages. Personally I do not program with UCSD-Pascal, but instead use Apple's Lisa Pascal and Apple's Macintosh Pascal (both of these may be considered supersets of UCSD-Pascal). I own an Apple ///, but I do not write code that takes advantage of its unique features.

USUS should (must?) extend its coverage to other Pascals since UCSD-Pascal and the p-System are no longer dominant in the microcomputer field. Other languages that I personally would like to know more about are Modula-2 and ADA. I worked with ADA in college and found it to be an excellent language for large projects. One language that must be avoided is C. Even though C has become the dominant language for microcomputers, it in my opinion, lacks a solid software engineering foundation. Pascal, Modula-2 and ADA are more in line with modern software engineering practices than C.

USUS may consider extending its coverage to object-oriented programming systems (OOPS). I've become very interested in Apple's Object Pascal and would like to see more coverage given to this language and others. OOPS are becoming more and more prevalent as exemplified by the new object-oriented Pascal compilers released by MicroSoft and Borland for the IBM PC.

Alex Kleider is definitely correct when he states that USUS's main problem is lack of member participation. In my opinion members of USUS do not participate because USUS has a lack of focus. The NewsLetter article, "Changes in USUS" states that USUS is focused on software portability. This is an admirable goal, but in my opinion is too restrictive. Portable software by its definition does not use unique features of machines. For example, any vanilla Pascal program can be compiled and run on the Macintosh, but the Macintosh version will lack windows, menus and mouse support. The majority of Macintosh users would never use a vanilla program on the Macintosh because the program is not "Macish". The same applies to the IBM PC world. PC programs that use only a subset of the PC's capability will not have wide acceptance in the PC marketplace.

I think USUS members will participate in USUS if they can be actively involved in worth-while projects. I would like to see USUS begin several non-trivial projects aimed at different machines. For example, I've been working on a cartography program for the past year for the Macintosh. I would very much like to finish this program, but my time is limited. If several Macintosh oriented members of USUS could put some time into a project of this nature, I think it could be finished in another year.

This same concept also applies to the IBM PC world. And if a major project is completed, it may even have commercial possibilities. I don't think USUS should cater to beginning programmers for any machine. It should instead aim its NewsLetter and library programs to experienced programmers. Plenty of magazines exist which cater to beginning programmers. USUS could never hope to compete against them.

- -

Sb: #COMMENT ON D. CRAIG
Fm: Hays Busch USUS Admin. 70260,306

I think David Craig makes some very valid points. But I do not agree with all of his conclusions. I would be concerned about pointing USUS exclusively at "experienced programmers" at the expense of "beginners", since I am not sure the "experienced programmers" could or would provide the numbers needed to keep the organization viable. (If USUS can remain viable even at its present numbers!)

I do agree that USUS should be more "machine oriented", and to that end I "pushed" the concept of Machine SIGS as Administrator. Unfortunately, with the exception of Frank's work with Apple and Mac, and to some extent with Ken Hamai in the TI 99/4a area, that concept did not take hold. So it may well have been incorrect.

I posted this in the USUS Members only section so it would remain USUS restricted. If any of the BofD or officers feel this should be given a wider distribution in an upcoming NewsLetter, I suggest that David be contacted for permission. I can provide his address to anyone so inclined via EasyPlex.

One final thought. David says he will send me an outline of his thinking on the Mac Cartographic program. Any of you who might be interested in working with him in developing this, should also let me know. 'David seems to be a very bright person and from what I have seen of his TIP program he is an excellent programmer. He seems to want to help USUS in any way he can.

- -

Sb: D. Craig's letter
Fm: Alex Kleider, Pres. 71515,447

Dear David,

Hays Busch shared the contents of your recent letter to him with some of the USUS officials. I would like to seek your permission to make it an open letter to be published in the USUS NewsLetter and also to make a few comments.

There is not likely to be any disagreement on most of the points you make. "In my opinion members of USUS do not participate because USUS has a lack of focus. " Don't forget that USUS policy is set by members who are willing to serve on the Board of Directors and there is an election due and we need candidates to stand for two positions. Would you be interested in being a candidate? We need people with ideas as to where to take the organization. Apart from soliciting candidates, my goal here is to make you see that the setting of focus is a "member participation" function and so blaming lack of participation on lack of focus is a circular argument!

I too would like to see programming projects get started and indeed there has been talk of such things on MUSUS (a USUS sponsored forum of CompuServe which I would encourage you to join). The problem however always gets back to the common problem: participation. Perhaps if you were to publicly announce your interest in further development of the cartography program, there might be others who would like to join you but it'll take someone to spearhead such a movement, namely you.

One area in which people will disagree is your comment about not catering to beginners. We feel strongly that we should remain a source for those beginners with UCSD p-System related questions since this group has nowhere else to turn. This function is not a big drain on resources so I don't think our disagreement on this issue is of great significance.

Thank you for your comments. Please let me know if we can publish our exchange since I think it would be of interest to the general membership.

Sincerely,

Alex Kleider, USUS President

- -

Letter from David Craig to Alex Kleider (November 18, 1989):

Dear Alex:

Thank you for your recent letter responding to my comments about USUS' current direction. You may do what you want with any of my letters I have sent to any USUS officers. I have meant none of them to be private to any one individual. My last letter to Hays Busch was sent to his Colorado address since all my letters to the La Jolla address end up in Colorado eventually. Since Hays is leaving USUS soon I will address my future correspondences to La Jolla.

You are correct in stating that USUS is its members. Without their cooperation USUS, like many other small organizations, is doomed to failure. Concerning your question about me becoming a Board of Directors member, I'm not sure I have the "right stuff" for such a position. I talked to Frank Lawyer a long time ago about USUS having an Apple Macintosh SIG. If such a group existed, I may be interested in working with it on a more official basis. My time is also limited. I work as a professional Macintosh programmer and when I'm at home I like to distance myself from programming. Forty to sixty hours a week of programming can become tedious no matter how much a person likes the job.

I will look into the specifics of MUSUS, but I've never been too thrilled with BBS services. I prefer the US Postal Service since it is cheap and convenient. My comments to Hays about my cartography program are meant to be public. If others in USUS are interested in a project of this nature I would be more than pleased to assist in the dissemination of my source code for MacCartog.

Sincerely,

David Craig

News From MUSUS

by Harry Baya

Musus is the current name of the USUS sponsored forum on the Compuserve Information Service (CIS). It serves as an international BBS for USUS. In prior years, especially when a lot hackers used Apple Pascal to program Apple]['s, we had a lot of message traffic on MUSUS. In the last year or two, for a number of reasons, the message traffic related to UCSD Pascal has greatly decreased. We are in the process of restructuring MUSUS to cover a wider range of interests. The USUS board approved renaming the forum the "Portable Programming Forum" last year and I hope to make that change and related restructuring "real soon now".

More than half the messages in recent months have dealt with Modula-2 and we will be promoting this area heavily when the restructuring is publicly announced.

For those of you who have put off using CIS for financial reasons there are programs available for IBM Dos, Macintosh and Amiga machines that make it very easy to do most of your work offline, which greatly reduces the cost. Contact me on MUSUS, by phone at 914-478-4241 or via my almost dormant BBS (The Pumpkin Patch) at 914-478-7359 if you would like more information about these programs.

We are actively seeking ways to be make MUSUS more attractive to the USUS community and your suggestions are welcome. One of my personal goals is to build a good on-line library of public domain source code in Pascal and Modula-2. Let me know if you want to help in this or other areas that would be beneficial to MUSUS.

Staff Meeting Minutes (January 9, 1990)

By Keith R. Frederick

Minutes of the staff meeting of USUS, Inc., held in room 1 of the MUSUS forum teleconferencing facility on the Compuserve Information Service January 9, 1990.

Present at the meeting were:

User ID	Name
71515,447	Alex Kleider (AlexK)
74076,1715	Felix Bearden (felix)
73007,173	William Smith (Wm)
73447,2754	**Henry Baumgarten (Henry) BoD**
70260,306	Hays Busch (AHB)
73760,3521	Keith R. Frederick (KeithF)
72767,622	Tom Cattrall (TomC)
73030,2522	Ron Williams (ronw)
76702,513	Harry Baya (Harry)
72747,3126	Bob Clark (BobC)

Because only one Board Director was present, a quorum couldn't be formed and the Board Meeting was changed to a Staff Meeting. Henry Baumgarten chaired the meeting. Topics dealt with were:

Helping out Hays Busch

Alex Kleider brought up the issue of getting official status for Felix Bearden to take over some of Hays' job. Since a quorum wasn't present and Hays needed the help with or without the Board's approval, a straw vote to approve Felix getting started was taken.

The vote was unanimously approved. Those who voted were: Hays Busch, Alex Kleider, Tom Cattrall, Harry Baya, and William Smith.

Voting in New Officers

Even though a quorum wasn't present to approve it, three positions were up for vote. The following were unanimously elected:

Stephen Pickett - Keeper of the Library
Tom Cattrall - USUS Newsletter Editor
Keith R. Frederick - Secretary of the Board of Directors
Alex Kleider, William Smith, Felix Bearden, Hays Busch, Tom Cattrall, and Harry Baya voted.

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

Arrangements with the publisher of JPAM to purchase the magazine subscriptions were made by Bob Spitzer. An agreement still has to be made with JPAM's editor about a USUS column. Alex Kleider declared that the issue of JPAM being a membership benefit was decided and all that is left are the implementation details. These details will be handled by himself, Felix Bearden, and Bob Spitzer.

All renewal letters sent out for the first quarter will include the JPAM benefit and the new renewel rate of $45.

The topic of giving current users a full year subscription was introduced, but William Smith declared that the original proposal was that members wouldn't get JPAM until they renewed at the higher rate. William also noted that USUS would lose money on those members who didn't renew their USUS membership but still received the free subscription. Both Henry Baya and Alex Kleider agreed with William, emphasizing that the original Board of Director's resolution must be followed.

Miscellaneous

The winners of the Board of Director's election will be known by the next meeting.

Harry Baya stated that he is willing to take responsibility for giving section 8 access to those taking on USUS responsibilities and would ask for approval at the next meeting. William Smith and Alex Kleider agreed with that.

Next Meeting

The Staff agreed to adjourn and meet again at 7 PM PST / 8 PM MST / 9 CST / 10 PM EST February 13th 1990 in Room 1 of the MUSUS conference facility.

The Staff Meeting was adjourned at 8:14:47 PM PST on January 9th, 1990.

Minutes submitted by:
Keith R. Frederick

Staff Meeting Minutes (Feb 13, 1990)

By Keith R. Frederick

Minutes of the staff meeting of USUS, Inc., held in room 1 of the MUSUS forum teleconferencing facility on the Compuserve Information Service February 13, 1990.

Present at the meeting were:

User ID	Name
71515,447	Alex Kleider (AlexK)
74076,1715	Felix Bearden (felix)
73447,2754	**Henry Baumgarten (Henry) BoD**
70260,306	Hays Busch (AHB)
73760,3521	Keith R. Frederick (KeithF)
72767,622	**Tom Cattrall (TomC) BoD**
73030,2522	Ron Williams (ron)
76702,513	Harry Baya (Harry)
72747,3126	Bob Clark (BobC)

Because a quorum couldn't be formed, the Board Meeting was changed to a Staff Meeting. Henry Baumgarten chaired the meeting.

Topics dealt with were:

Election Results

The winners of the election were Stephen Pickett and Tom Cattrall.

Meeting Attendance

Alex Kleider suggested that the minutes should indicate both who showed up during the meeting and when. This is to provide more information to the members about candidates for future elections.

Tom Cattrall suggested that making a rule, such as if a board member missed two consecutive meetings s/he would be off the board, might provide incentive for board members to attend. Henry Baumgarten agreed with this measure.

Alex further suggested that the next newsletter indicate the current attendance problem.

Felix Bearden's Proposal

Felix stated that his proposal was to split the Administrator's job into two. One for membership responsibilities and the other for member services. Felix made clear that he already volunteered for one of the roles but that he couldn't do everything.

Because the proposal was posted just before this meeting, Henry Baya suggested that everyone read it and prepare to amend or take action on it next month.

Alex Kleider mentioned that Felix work out his proposal with the people involved on a one-to-one basis and if everyone involved agrees we will give it our blessings. Alex also stated that this matter can be done outside of meetings.

Hays Busch commented on the proposal, indicating that the membership database and financial monthly journal should be closely coordinated because they are required for USUS to run properly. Felix Bearden agreed.

New Meeting Time

In order to accommodate more members, changing the monthly Board (or in this case Staff) meetings was discussed. While a permanent change was not made official, a consensus was made to meet on the second Wednesday of each month at 6:30 pm PST.

Next Meeting

The Staff agreed to adjourn and meet again at 6:30 PM PST / 7:30 PM MST / 8:30 CST / 9:30 PM EST March 14, 1990 in Room 1 of the MUSUS conference facility.

Minutes submitted by:
Keith R. Frederick

Prez Sez

by Alex Kleider

As previously mentioned in this irregular column, USUS appears to be weathering the crisis of dealing with Hays Busch's resignation and the switch to a new NewsLetter editor but the problems are not all solved and now is no time for complacency.

The last two scheduled meetings of the USUS BoD turned out to be "non meetings" for lack of quorum. Items were still discussed by those present but this is no way to run an organization. Every member of the Board must be strongly encouraged to attend each meeting and each member of USUS should consider lobbying the Board Members to attend and to represent their interests. It is only based on their performance at these meetings that you as members can judge how to vote next time. Pay attention. USUS depends on it.

One of the problems currently causing great concern is that Felix who has taken over much/some of Hays' work has more than he can handle. He needs help and if it isn't forthcoming we may end up with having our records in shambles again as they were before Hays took over. Other functions such as IBM/compatibles SIG chairmanship still remain empty.

Hopefully by now those of you who have recently joined or renewed are already getting JPAM as a membership benefit. Let your Board Members know if this is appreciated. They deserve some credit for their positive actions as well.

We Get Letters

Keith Frederick
Call for Apple IIGS Users

I was wondering if you could put in a plug in the newsletter for me indicating that I'm interested in talking with other users of Pecan's Power System for the Apple IIGS. Pecan doesn't seem to have any interest in the Apple IIGS, so I figure others who have a GS and the Power System may be interested in trading some notes and ideas. Go ahead and mention my phone number (206-285-1576) and address, both postal and email if you have room.

Keith R. Frederick
525 West Prospect St.
Seattle, WA 98119
scalawag@blake.acs.washington.edu

Randy Bush
Update on Modula 2 Standard Draft Proposal

The Draft Proposal is out. The US will ballot the end of this month. The Brits have already voted nay, as it is not seen as ready for prime time. WG13 meets in early June to review.

Peter Perchansky

Peter in a letter to Alex Kleider wrote to say that he is no longer on Compuserve and then continued with:

Even though I have stopped using CIS, I have not stopped programming in Modula-2. I have also, hopefully to your delight <grin>, been putting pressure on JPI to flag extensions in their TopSpeed Modula-2 product - I am still concerned about portability issues.

My recent Modula-2 library project is the development of the ability to use dynamically allocated strings: ie. strings that can vary in size, and only use what memory is necessary - strings whose size does not have to be declared before compilation time.

I hope to have it done in time to get published in the next edition of USUS.

James P. Harding

I was going through a book store called Computer Literacy in the city of Sunnyvale where I ran into a great UCSD Pascal book. The title is *Programming Language Translation a Practical Approach* by Patrick D. Terry. It provides an in depth study of concurrency programming (UCSD Pascal parallel processing) and data abstraction which I am interested in. Also Mac (LONGINT) long integer and object code.

I would like to see UCSD Pascal become an Object-oriented language. There was a good article in Byte July 1989 titled *Clash of the Object-Oriented Pascals* and I would like UCSD Pascal to move into this area.

It was funny how I found the address to get the source code that is in the book. The book had only Rhodes University, Grahamstown as the address to send for disks. UCSD Pascal is so often mentioned in the book that I tried looking up the author's name in the March 1989 USUS Newsletter and found it there.

There is a nice plug in the book for the UCSD Pascal User's group USUS and it gives the P.O. Box address at La Jolla to get the source code of the UCSD Pascal compiler release 1.3 for a project that could be worked on. Would you please plug this book in the next newsletter?

Consider it plugged. James' review follows.

Programming Language Translation, a Practical Approach by Patrick D. Terry Addison Wesley Publishing Company

Reviewed by James P. Harding

The book begins with an overview of the translation process and the constituent parts of a compiler. Features of assembly language are covered and an assembler / interpreter allowing for conditional and macro assembly is developed.

Formal syntax theory and parsing are introduced. A recursive descent compiler / interpreter for a simple Pascal-like source language is constructed producing code for a hypothetical stack based computer.

Extensions to the simple language are introduced, including procedures and functions, and constructs to handle the increasingly important areas of data abstraction and concurrency. The compiler / interpreter system grows in complexity and usefulness until the final product enables the development of quite sophisticated programs.

Address:

P.D. Terry
Computer Science Department
Rhodes University
Grahamstown 6140
South Africa

Administrator Sez
by Felix Bearden

It is becoming more and more apparent what a good job Hays Busch did as Administrator. The fact that I have not called him more often is a testimony of his organization of the job rather than either my reluctance to call or his reluctance to help. So Again:

THANK YOU HAYS!!!!!

When the board decided to adopt the "Journal of Pascal, Ada, and Modula-2"(JPAM) as the "official" journal of USUS it also decided that all memberships expiring after December 31, 1989 would renew at the new rates and be eligible for the subscription. The board rightly felt that current members (members expiring on or before December 31) could not be supplied the journal because of the cost to USUS. If you are a current member and would like to receive JPAM please send your responses to the following statements to me.

[ ] I would like a subscription only if it were free.

[ ] I would be willing to pay no more than an additional $10 for the subscription.

[ ] I would be willing to pay an additional $20 for the subscription.

Send a post card to the USUS address, Attention: Administrator, or post a message on MUSUS to me, or leave a message at (404)923-4825.

We are about to learn this job. If I'm late responding or don't do what you expect me to, let me know, then remind me, and if that doesn't work, let me know and remind me. Meanwhile, I'm going to have a "cooler" - and thanks for your support.

Felix

Treasurer's Reports
by Robert E. Clark, Treasurer

January 1990

Bank Balance	$6,771.34	12-31-89
Income - January 1990		
Dues:		(new/renew)
Student	0.00	0/0
General	170.00	1/4
Professional	100.00	0/1
Institutional	0.00	0/0
Other Income:		
CIS	0.00	
Library fees	45.00	
Publications	0.00	
PowerTools	0.00	

Total Income:	$ 315.00	
Expenses - January 1990		
Administrator:		
CIS	56.19	
Telephone	8.04	
Postage	120.55	
Printing	162.11	
Photocopies	4.60	
Supplies	48.01	
Other:		
Bank charges	2.00	
Newsletter	0.00	
Mail from La Jolla	6.00	
USPS (stamps)	50.00	
Reimbursements	0.00	

Total Expenses	$ 457.50	
Bank Balance	$ 6,628.84	1-31-90

February 1990

Bank Balance	$6,628.84	1-31-90
Income - February 1990		
Dues:		(new/renew)
Student	0.00	n/a
General	280.00	n/a
Professional	100.00	n/a
Institutional	0.00	n/a
Other Income:		
CIS	30.11	
Library fees	0.00	
Publications	0.00	
PowerTools	0.00	

Total Income:	$ 410.11	
Expenses - February 1990		
Administrator:		
CIS	116.20	
Telephone	45.84	
Postage	29.37	
Photocopies	9.20	
Supplies	13.94	
Other:		
Bank charges	2.00	
Newsletter	334.81	
Mail from La Jolla	16.57	
Reimbursements	0.00	

Total Expenses	$ 557.93	
Bank Balance	$ 6,471.02	2-28-90

When Print Heads Go Bad

by Tom Cattrall

A few months ago I noticed that my NEC P2200 24 pin printer had developed a light streak while printing some graphics. At first I thought it was due to paper positioning problems or a bug in my code but further investigation showed that the top pin in the printhead wasn't working any more.

A trip to the local dealer where I had purchased the printer wasn't very helpful. They didn't really want to work on it and didn't know what the cost might be to repair it. The last time they had ordered a printhead from NEC it was over $400. This was for a larger printer but as my entire printer cost under $400, I said no thank you and left. A rather unpleasant experience.

Remembering some ads in Computer Hot Line for printhead repair services, I dug out a copy and ended up calling an outfit by the name of Impact Printhead Services in Austin Texas. The woman I talked to was very helpful and told me that it would cost $65 to repair the P2200 printhead. So I got an RMA number but then waited for a couple of weeks because I needed the printer and didn't want it to be out of action for several weeks. Finally things quieted down and I sent the printhead off with a check for $65.

Contact Information

Impact Printhead Services
8701 Cross Park Drive, Suite 101
Austin Texas 78754
(512)832-9151
(800)777-4323

Be sure to call for an
RMA number first.

The printhead is easy to remove on most printers. On the P2200 you just unfasten a large clip, slide the printhead out of a guide, and disconnect a slide-on card edge connector from the head. The whole process took about a minute the first time I did it and much less after I knew how.

A week later I came home to find a UPS package on my porch. The entire trip from Oregon to Texas by US Postal Service, the repair, and the return trip via UPS had taken just a week. I was amazed.

I replaced the printhead and it worked flawlessly. Impact should be congratulated. Such fast and good service is not all that common these days.

To summarize what they offer:

Repair of printheads: Prices of 9 pin heads are typically $35 to $50. A large range of printers and models is supported. 24 pin heads seem to be in the $65 to $90 range. The highest cost in their list is $145 for a Panasonic 1524. A few cost $125 and all other prices are less than $100.

Sell refurbished heads: Prices about 50% higher than repair. Only a few printer models available.

Factory new heads: Cost more than refurbished heads. More selection than refurbished but nowhere near as many as can be repaired.

As an example for a more popular printer, my 9 pin Okidata 93 printer would cost $47 to repair, $65 to buy a refurbished head ($9.60 rebate for an old printhead), and $115 to buy a factory new printhead ($20 rebate for old head).

Repairs are guaranteed for 6 months and if they can't repair a printhead they return it at no charge.

In summary, if your printer fails because of a bad printhead, Impact provides a quick and relatively inexpensive way to get it working again. And with no hassles.

Real->Integer->Real

by Harry Baya

The following is from an exchange of messages on the Compuserve Forum. The Code referred to here is shown on the next page.

From Harry Baya to ALL:

I am writing a program to read a P-system data file and write out a "portable" data file that can be read by a companion program in a different system (TML Pascal on a Macintosh). The TML pascal will decode the values and write them out in its own format in the "target" file. This process will be used for a year or two in several offices.

[P-System file] ->[portable file] -> [Mac. File]

I can handle integers and strings ok, but have yet to find a good way to handle reals. The real numbers take up 4 bytes in the source file and I am willing to use 5 or 6 bytes for each real in the portable file. This would be easier if I had a reasonable way to convert:

(a) real to longinteger
and
(b) longinteger to real

Would anyone like to suggest a method for converting the data or a way to convert between real and longinteger?

My current solution to the problem is shown below, but I am not happy with it.

Reply from Tom Cattrall:

Unless you want to get into the actual real implementations on the 2 machines, what you've got is probably as good as anything else.

You could do a real to long integer by adapting my real to string algorithm presented in NL 15. Unless you really want long integer, your method is probably preferable.

Reply from Harry:

Looking back I think it would be more straight forward to simply take the square root of the number rather than dividing by 306. This is clearly a hacker's (as contrasted with the expert) program rather than a state-of-art process. That's ok, the others like me will understand it and, with luck, some of the better programmers will suggest much better solutions.

Reply from Tom:

Rather than wasting time on a square root, I'd probably scale the value to be : 0.0 <= X < 10000.0 and then do the following:

```
part1 := TRUNC(X);
part2 := TRUNC(10000.0 * (X - part1));
```

Thus, part1 gives highest 4 decimal digits, and part2 gives another 4 although not all 4 of part2 will be accurate in all cases. The sign of the number can be passed as the sign of part1 since part1 is < 10000.0.

The exponent (scale factor) can be passed in 8 bits so that you are using 5 bytes instead of 6.

Reply from Harry:

Tom, Thanks for the suggestion. I prefer your method to mine. I don't know that I can pass the the exponent (scale factor) as 8 bits unless I put it there as part of a packed record... and I'm suspicious that the packing might not travel well. It's worth trying and I will.

```
{  r2i - Real to integer, uses three
          integer values
   i2r - Integer to real, uses three
          integers to convert back to
          real
}
procedure r2i( r1:real; var div306,
                 rem306, log10: integer);
 var logr, r2,r3,r4, r5, r6  : real;
      isminus : boolean;
      tenfactor :integer;

 begin
   isminus := (r1 < 0.0);
   if isminus then r1 := -r1;
   logr := log(r1);
   log10 := round(logr);
   tenfactor := 6 - log10;
   if (tenfactor > 0 ) then
     r2 := r1 * pwroften( tenfactor)
   else
     r2 := r1 / pwroften(-tenfactor);;
    r3 := r2/306.0; {minimize error}
    div306 := round(r3);
    r4 := div306;
    r5 := r4 * 306.0;
    r6 := r2 -  r5;
    rem306 := round(r6);
    if isminus then div306 := -div306;
 end;


procedure i2r( var r1:real; div306,
                 rem306, log10: integer);
  var r4, r2, r6 : real;
        isminus : boolean;
begin
  isminus := div306 < 0.0;
  if isminus then div306 := -div306;
  r4 := div306;
  r6 := rem306;
  r2 := (306.0 * r4) + r6;
  tenfactor := 6 - log10;
  if (tenfactor > 0 ) then
    r1 := r2 /pwroften( tenfactor)
  else
    r1:= (r2 * pwroften(-tenfactor));
  if isminus then r1 := -r1;
end;


begin
  repeat
    write  ('Enter number : ');
    readln(numr);
    inum := 0;
    if (abs(numr) < 1000.0 ) then
      begin
        inum := round(numr);
      end;
    if (inum <> 99 ) then
      begin
        writeln;
        write('Entered:',numr:12:4);
        r2i( numr, div306, rem306,
               log10);
        i2r( numr,  div306, rem306,
             log10);
        writeln('returned:'numr:12:4);
     end;
   until (inum = 99 );
end.
```

Bug Box by Tom Cattrall

Last summer I was rereading my favorite book on Modula II: Modula-2: Discipline & Design by Arthur Sale. He has a separate section in the back of the book titled "Other Modula-2 Features" where he discusses features of the language that should be avoided because they contribute to error prone programs.

The REPEAT statement is the first statement mentioned. The reasons given didn't seem especially convincing to me. Primarily he feels that the REPEAT statement isn't simple to use with loop invariants which are covered in the main part of the book.

As an experiment, I went through a module that I had recently coded and only tested slightly before setting it aside. It was about 700 lines long. I searched for REPEAT statements and tried replacing the code with WHILE statements.

It came as a surprise that in each case the resulting code came out somewhat more clear. An even bigger surprise was that I found several bugs in the process. Maybe he has something there.

A few weeks later, a coworker went out to the field to fix some problems that were occurring in a previously installed system. He reported back that many of the problems he found and fixed involved REPEAT statements. I hadn't discussed my experience with him so this was a completely independant observation.

I now regard the REPEAT statement with suspicion. Has anyone else had similar experiences with the REPEAT statement?

Computing Large Factorials

by Tom Cattrall

```
169! =
                                             42,690,680,090,047,052,
      749,392,518,888,995,665,380,688,186,360,567,361,038,491,634,111,
      179,775,549,421,800,928,543,239,701,427,161,526,538,182,301,399,
      050,122,215,682,485,679,075,017,796,052,357,489,455,946,484,708,
      413,412,107,621,199,803,603,527,401,512,378,815,048,789,750,405,
      684,196,703,601,544,535,852,628,274,771,797,464,002,689,372,589,
      486,243,840,000,000,000,000,000,000,000,000,000,000,000,000,000
```

The May 1989 issue of the ACM SIG Small newsletter had an article that discussed computing the factorials of large numbers (n!). The author of that article discusses a few sketchy details of how he went about the task of handling the large numbers that result. Basically, he seems to have coded up 68000 assembly routines to handle long binary numbers 2 bytes at a time. They worked but are very slow. To calculate 20365! took around 24 hours. But to convert the binary result to ascii for printing took 32 days. When I read that I figured that another approach would be faster.

The method I used was to do all computations in a form of binary coded decimal (BCD). If you want to multiply 1234 * 5678 the operations look like:

```
   1234
   5678
   ----
   9872
  8638
 7404
6170
-------
7006652
```

The numbers could be represented as digits in arrays of 0..9 so that the values would be stored one digit per array element with the least significant digit stored in the first element. The numbers from the example above would be stored as:

```
n1[1] = 4,  n1[2] = 3,  n1[3] = 2,  n1[4] = 1
n2[1] = 8,  n2[2] = 7,  n2[3] = 6,  n2[4] = 5
```

Then to do arithmetic, use for loops to deal with the individual digits in the same way you would do it by hand. Thus to add n1 + n2 giving n3:

```
FOR i := 1 TO 4 DO
  n3[i] := n1[i] + n2[i];
  IF n3[i] > 9 THEN
    n3[i+1] := n3[i+1] + n3[i] DIV 10;
    n3[i] := n3[i] MOD 10;
    END;
  END;
```

The above code fragment isn't complete or general but it illustrates the basic idea. Printing out the result is trivial:

```
FOR i := 4 TO 1 BY -1 DO
  WriteCard( CHR( n3[i] + ORD('0') ) , 1);
  END;
WriteLn;
```

You can see that this approach is much faster than that described in the article, if for no other reason than that the conversion to ascii for printing is going to run in a matter of seconds rather than days.

For doing factorials however, we need a somewhat more complicated loop to multiply. Rather than code up a general purpose multiply routine to handle 2 large values in the format above, I made some simplifications in the interest of gaining speed.

First, I decided to keep the multiplier as a normal integer and save the longer and slower format for the result. For a 16 bit integer, this means that the code will be limited to 32767!, a very large number.

The second improvement is to keep more than one digit per array entry. Thus, each array entry contains 0..99 rather than 0..9. By using 0..99, we cut the storage requirements in half, and because each step will operate on 2 decimal digits, the time required will be roughly half. The printing of the result is slightly more complicated because we need to split each array entry into n3[i] DIV 10, and n3[i] MOD 10 and print the 2 digits separately. The time penalty for doing this is minor though compared to the savings we get.

The resulting multiply loop can be seen in the final listing given on the next page. The inputs are the current value of the multiplier (m), and 2 large numbers: the input value and the result. nbWords says how many array entries are currently in use so that we save time by not traversing unused zero entries when doing smaller numbers. Because the routine uses one array and stores into a second, the caller allocates 2 arrays and alternates calls so that the array that held the result from the last call is used as input for the next call. This saves having to copy the result array to the input array before doing the next call.

The main program loop and the multiply procedure were easy to code up and get running. Formatting and printing the answer, which should have been an afterthought, caused me quite a bit of grief. I just threw some code together so I could see the results. Sure enough, all of the digits were printed, but the formatting was sloppy. What I wanted to see was nicely aligned rows like those in the number at the top of the previous page. What I got was a poorly aligned mess that showed that I had better take the formatting code seriously too. I made a couple of attempts at patching it but things didn't get better, just changed symptoms.

So I started over and actually thought about what I needed and how to do it. The resulting code first computes how many lines of output we'll have. Then the number of leading zeroes for the first row is found by taking the difference between how many digits we think we'll get and how many digits fit if all the lines were full. If there is only one row I don't bother with any alignment and nbLeadingZeroes is set to 0.

Then the leading zeroes are output using the WriteDigit routine. This is followed by a loop to output all of the digits, 2 per word. The WriteDigit routine handles the suppressing of leading zeroes, the line ending and left margin, and uses PrintComma to put out a comma at the proper time. The number of digits remaining to be printed is passed around so that WriteDigit and PrintComma know when to take action.

The result of all this is a routine that can compute the factorial of large integers in a much shorter time than the binary approach described in the ACM article. The time required to compute 169! is 0.5 seconds on a 16 Mhz 68020. Trying a few other values gives the following table:

N	Time(Secs)	Digits In Result
169	0.5	305
500	6.3	1135
1000	29.8	2544
2000	136	5513

The time seems to be order N squared which is what you'd expect for this code. The number of digits in the result seem to be order N.

I changed this routine to Pascal and ran it on a MicroVax III using 32 bit integers and 4 digitsPerWord. The time to compute 20365! was around 3 hours. On my machine the time was about 5 hours. The result has 78909 digits.

What good is all of this? Well, I don't know of any good use for the actual values of large factorials. The method of doing arithmetic on large numbers is useful in general. An entire large integer package can be coded using this general approach. I found the exercise interesting for two reasons.

First, I suspected that this approach would show an improvement over the speeds reported in the ACM article. As can be seen, the speed difference is very large.

Second, I learned that the "trivial" detail of formatting the output deserved just as much care as the algorithm that I set out to work on in the first place.

```
MODULE Factorial;
(*
    This program prompts for an integer and then computes
    the factorial of that integer. The size of the result
    is limited  by the CONST nbDigits defined in the program.

    It allows for very large results to be computed by doing
    arithmetic using a form of binary coded decimal with one
    or more digits packed into a word. See the Multiply
    PROCEDURE for details.
*)


FROM InOut IMPORT WriteLn, WriteString, Write, WriteCard,
                  ReadCard;

CONST
       nbDigits     = 30000;    (* max number digits in answer *)
       digitsPerWord = 2;       (* how many digits we pack per word *)
       byteValue    = 100;      (* 10^digitsPerWord *)

       digitsPerLine = 48;       (* how many digits in output line *)
       leftMargin    = '    ';  (* left margin for output line *)

       CR            = 13;   (* ascii carriage return character *)

TYPE
      aDigit       = [0..9];
      aDigitCount  = [0..nbDigits];
      aLargeNbIndex = [1 .. (nbDigits DIV digitsPerWord)];
      aLargeNumber = ARRAY aLargeNbIndex OF CARDINAL;

VAR
    i        : aLargeNbIndex;
    number   : CARDINAL;
    nbWords  : aLargeNbIndex;
    f1,
    f2       : aLargeNumber;



PROCEDURE PrintComma( n : aDigitCount ; leadingZero : BOOLEAN);
(*
    Checks to see if it is time to print a comma to
    separate the digits into groups of 3.
    n is the number of the digit just printed, leadingZero
    says whether any non zero digits have been printed yet.
*)

BEGIN
  IF (n >= 3) AND (n MOD 3 = 1) THEN
    IF leadingZero THEN
      Write(' ');
    ELSE
      Write(',');
      END;
    END;
END PrintComma;

PROCEDURE WriteDigit( VAR leadingZero    : BOOLEAN;
                      VAR n              : aDigitCount;
                          d              : aDigit);
(*
    Prints the single digit d. leadingZero is TRUE until
    the first nonzero digit is printed. n is how many
    digits remain to be printed.
*)

BEGIN
  IF (d = 0) AND leadingZero THEN
    Write(' ');
    PrintComma(n, leadingZero);
  ELSE
    Write( CHR( ORD('0') + d ));
    leadingZero := FALSE;
    PrintComma(n, leadingZero);
    IF (n > 0) AND ((n MOD digitsPerLine) = 1) THEN
      WriteLn;
      WriteString( leftMargin );
      END;
    END;
  n := n - 1;

END WriteDigit;



PROCEDURE WriteAnswer( VAR f : aLargeNumber);
(*
    Prints out the result. Format is digits in groups
    of 3 separated by commas.
*)

VAR
    i                : CARDINAL;
    nbLines          : CARDINAL;
    nbLeadingZeroes  : CARDINAL;
    nbDigits         : aDigitCount;
    leadingZero      : BOOLEAN;

BEGIN

(*   First compute how much we must print   *)

  nbLines := (digitsPerWord * nbWords + digitsPerLine - 1) DIV
             digitsPerLine;
  IF nbLines > 1 THEN
    nbLeadingZeroes := nbLines * digitsPerLine -
                       digitsPerWord * nbWords;
    nbDigits := (nbLines * digitsPerLine);
  ELSE
    nbLeadingZeroes := 0;
    nbDigits := digitsPerWord * nbWords;
    END;

  WriteLn;
  WriteCard( number, 1); WriteString('! = ');
  WriteLn;  WriteString(leftMargin);

  leadingZero := TRUE;              (* output leading blanks *)
  FOR i := 1 TO nbLeadingZeroes DO
    WriteDigit( leadingZero, nbDigits, 0);
    END;

(* the following FOR loop code depends on digitsPerWord = 2 *)

  FOR i := nbWords TO 1 BY -1 DO
    WriteDigit( leadingZero, nbDigits, (f[i] DIV 10  )        );
    WriteDigit( leadingZero, nbDigits, (f[i]         )  MOD 10);
    END;
  WriteLn;
END WriteAnswer;


PROCEDURE Multiply(     m    : CARDINAL;
                VAR in,
                    out  : aLargeNumber);
(*
    Multiply the large number "in" by the value "m" and place
    the result into the large number "out".
*)

VAR
    i, j   : CARDINAL;
    p      : CARDINAL;

BEGIN
  IF nbWords + 8 >= nbDigits DIV digitsPerWord THEN
    WriteString(' Overflow at N = ');
    WriteCard(m , 8);
    WriteLn;
    HALT;
    END;

  FOR i := 1 TO nbWords + 8 DO
    out[i] := 0;
    END;

  FOR i := 1 TO nbWords DO
```

```
    p :=  m * in[i];
    j := i;
    WHILE p + out[j] >= byteValue DO
      p := p + out[j];
      out[j] := p MOD byteValue;
      p := p DIV byteValue;
      j := j + 1;
      END;
    out[j] := out[j] + p;
    END;

  nbWords := nbWords + 8;
  WHILE out[nbWords] = 0 DO
    nbWords := nbWords - 1;
    END;

END Multiply;



BEGIN
  WriteString('Enter value for factorial : ');
  ReadCard( number );
  WriteLn;
  WriteString('Computing ');
  WriteCard( number, 1);
  WriteString(' Factorial');
  WriteLn;

  nbWords := 1;
  f2[1] := 1;

  FOR i := 2 TO number DO
    IF ODD( i ) THEN
      Multiply( i, f1, f2 );
    ELSE
      Multiply( i, f2, f1 );
      END;
    IF i MOD 100 = 0 THEN
      WriteCard(i, 8); Write(CHR(CR));
      END;
    END;

  IF ODD( number ) THEN
    WriteAnswer( f2 );
  ELSE
    WriteAnswer( f1 );
    END;

  WriteLn;

END Factorial.
```

Apple Macintosh Dialog Phrase Manager

David Craig
9939 Locust # 4013, Kansas City, MO 64131

Introduction:

The Macintosh Toolbox provides several powerful routines that create and manipulate dialogs. Dialogs are special windows that can contain text, various types of buttons, editable text areas, and graphics. But with this power comes a lot of programming difficulties. Dialogs are easy to create using one of the many Macintosh resource editors, but handling them from a program can become complicated. Many times a programmer only needs a dialog to display some text for the user and have at most several buttons that define different choices. For example, when a user quits a Macintosh program a dialog usually appears asking if the active document should be saved. Associated with the dialog are two buttons labeled "Save" and "Don't Save".

In order to simplify both the creation and manipulation of dialogs that contain only an icon, text, and buttons I created the Dialog Phrase Manager unit. The programmer need not be concerned with defining the location and dimensions of the dialog. The Dialog Phrase Manager automatically centers the dialog just below the menubar and makes the dialog box fit both the text phrases and the button list. This unit is written in MPW (Macintosh Programmer's Workshop) Pascal and should easily be portable to other Macintosh Pascal compilers.

About the Dialog Phrase Manager:

This unit allows Macintosh programmers to define, display, and handle special Macintosh dialogs. These dialogs consist of an icon in the upper left, textual phrases in the middle, and a vertical list of buttons in the right portion of the dialog window. The dialog is defined as a standard "STR#" resource. The first line contains the Header fields, the following lines contain the dialog textual phrases, and the last lines contain the button names. A default button can also be defined so that the dialog user can select it using the keyboard. Several standard phrase arguments exist that will automatically be replaced by other phrases. For example, the phrase "Hello ^1" will be expanded using the current value of the argument phrase ^1. If ^1 is defined as "David", then the phrase becomes "Hello David". Several arguments exist with the names "^1", "^2", ... , "^12".

Having the dialog definition defined as a Macintosh "STR#" resource makes modifications to it very easy through either the resource text source file or through a Macintosh resource editor such as Apple's ResEdit desktop application.

Using the Dialog Phrase Manager:

Before using the Dialog Phrase Manager the standard Macintosh managers must be initialized (your application should do this). The first routine to call is the initialization routine PM_Initialize. The last routine to call is PM_Terminate. Both return an error result which you should test. PM_VersionInfo returns a string containing the version number and compilation date and time of this unit. PM_ShowDialog is the major routine of this unit. It displays the dialog and handles the user's interaction with it. PM_SetArgAlert sets up phrase arguments. This provides a simple method of customizing dialog messages. PM_BeginWait displays a non-modal dialog that does not contain any buttons and returns immediately to the caller. Call PM_EndWait to remove the dialog from the screen. These two calls are useful when you wish to inform the user of an activity that requires some period of time. PM_GetPrivateCursor extracts the cursor data for the various special cursors that this unit can use. These cursors are normally accessed from the dialog definition's cursor fields. To print the text of a dialog use PM_PrintDialog. This displays a dialog informing the user that the dialog is being printed. This displayed dialog is specified by you by its dialog resource ID. When a dialog is displayed the user can select the buttons in several different ways. Pressing a button with the mouse selects the button. Pressing the keyboard Return or Enter keys selects the default button (surrounded by a dark outline). Pressing a keyboard function key (F1-F15) selects the corresponding button.

E.g., if F1 is pressed, the top button (#1) is selected. Pressing F2 selects the next-to-top button (#2), etc... Note that at most 15 buttons may exist within a dialog. If you include more buttons in the dialog definition than allowed an error is returned.

Sample phrase dialog definition:

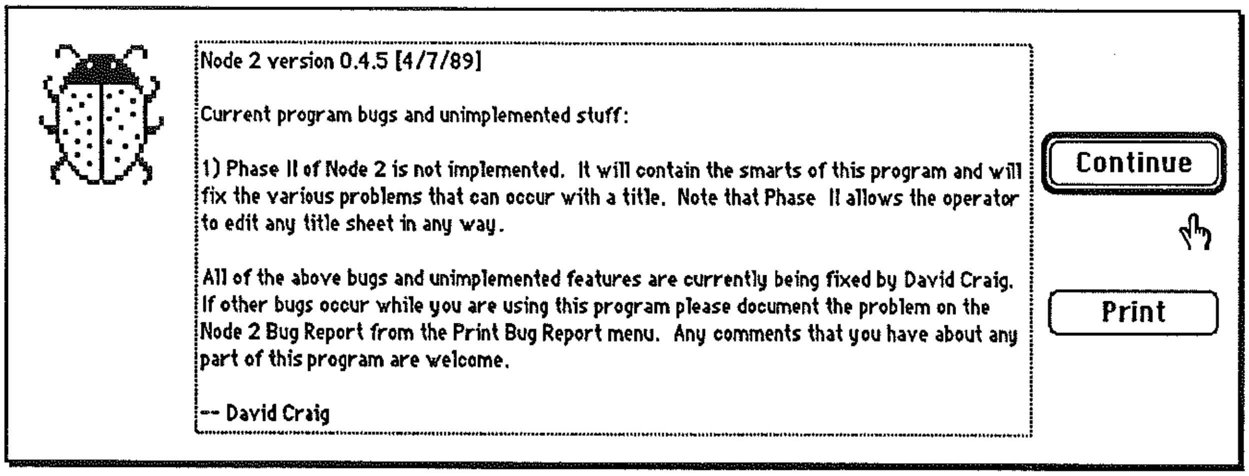
This definition is for a dialog that contains an icon, several lines of phrases, and two buttons. The definition is implemented as a Macintosh "STR#" resource and is written for the MPW development environment's Rez tool. All button names occur at the bottom of the definition and begin with a "~" character. The default button contains a "!" character after "~".

```
resource 'STR#' (143, nonpreload) {{
    /* iconID/IColor/FKind/FSize/FColor/inCursorID/outCursorID/parms */
    "128/red/geneva/9/black/32000/0/65",

    /* phrases */
    "Node 2 version ^1 [^2]\n\n",
    "Current program bugs and unimplemented stuff:\n\n",
    "1) Phase II of Node 2 is not implemented. It will contain the smarts of this program ",
    "and will fix the various problems that can occur with a title. Note that Phase II ",
    "allows the operator to edit any title sheet in any way.\n\nAll of the above bugs and ",
    "unimplemented features are currently being fixed by David Craig. If other bugs occur ",
    "while you are using this program please document the problem on the Node 2 Bug Report ",
    "from the Print Bug Report menu. Any comments that you have about any part of this ",
    "program are welcome.\n\n-- David Craig",

    /* buttons */
    "~!Continue", /* 1 (default button) */
    "~Print"      /* 2 */
    }};
```

Sample dialog display:


Node 2 version 0.4.5 [4/7/89]

Current program bugs and unimplemented stuff:

1) Phase II of Node 2 is not implemented. It will contain the smarts of this program and will fix the various problems that can occur with a title. Note that Phase II allows the operator to edit any title sheet in any way.

All of the above bugs and unimplemented features are currently being fixed by David Craig. If other bugs occur while you are using this program please document the problem on the Node 2 Bug Report from the Print Bug Report menu. Any comments that you have about any part of this program are welcome.

-- David Craig

That's all, Folks ...

Programming the Macintosh

by David T. Craig
736 Edgewater, Wichita, Kansas 67230
(October 31, 1988)

INTRODUCTION

Writing programs for the Apple Macintosh is very different than writing programs for conventional computers. The Macintosh's use of windows, menus, and the mouse give it a distinctive look and feel. Conventional computer programming is based upon reading character keystrokes, writing characters to the screen, processing the keystrokes, and repeating this ad infinitum. The Macintosh extends this programming methodology in a more general manner. Instead of waiting for keystrokes the Macintosh waits for events. Instead of writing characters to the screen the Macintosh draws images in windows.

THE TOOLBOX

The Macintosh software foundation rests upon the User Interface Toolbox managers, a set of about 500 routines implemented in 68000 assembly language that reside in the Macintosh ROM. These routines implement windows, menus, dialog boxes, and the other elements of standard Macintosh applications. The Toolbox's main managers are QuickDraw (provides the foundation for windows and draws everything on the screen), the Event Manager (handles event detection), the Window Manager (creates windows and handles user window manipulation), the Menu Manager (displays the menubar and the pull-down menus), and the Resource Manager (handles all access to Macintosh resources)

THE MAIN EVENT LOOP (MEL)

Every Macintosh program contains a Main Event Loop (MEL). This loop waits for events to occur and handles the events. A Macintosh event is any activity which causes the application to do work. The Event Manager supports 16 different event types which describe such activities as pressing the mouse button, pressing a keyboard key, and inserting a disk into a drive. The Pascal code for a typical MEL appears as

```
VAR myEvent : EventRecord; { Macintosh event record }
REPEAT  { fetch an event from the Macintosh }
  IF GetNextEvent(EveryEvent,myEvent) THEN
    BEGIN  { handle the event as needed }
      CASE myEvent.What OF
        MouseDown   : handle mouse button down event
        MouseUp     : handle mouse button up event
        KeyDown     : handle key down event
        AutoKey     : handle key repeat event
        ActivateEvt : handle window activate event
        UpdateEvt   : handle window update event
        DiskEvt     : handle disk insertion event
        DriverEvt   : handle driver event
        NetworkEvt  : handle network event
        Appl1Evt    : handle application-defined event
        Appl2Evt    : handle application-defined event
        Appl3Evt    : handle application-defined event
        Appl4Evt    : handle application-defined event
      END;
    END;
UNTIL done;
```

MACINTOSH RESOURCES

Every Macintosh program file consists of a set of resources. These resources contain descriptions of various pieces of the program. For example, menus are stored as resources. Resource compilers convert textual descriptions of program resources into resource object code and link the resource code into the application code. For example, the resource description for a menubar which has two menus titled File and Edit would read:

```
type MENU          type MENU
,128               ,129
File               Edit
Quit               Cut
                   Paste
```

The Toolbox Resource Manager provides calls that extract resources from applications. Resources make modifications to application phrases and menus very easy. Using a resource editor, such as Apple's ResEd, a person who knows nothing about the internal coding of an application can easily change the application menu phrases.

MACINTOSH WINDOWS and PRINTING

All displays for the Macintosh occur within windows, rectangular areas on the screen which act like pieces of paper. The Window Manager allows window creation, deletion, movement, and resizing. Windows can also overlap one another. Drawing occurs within a window when the application receives an update event that tells the application to redraw a window's contents. Printing on the Macintosh is also very easy and powerful. The Toolbox Print Manager provides an off screen window which applications draw into. Low–level printer drivers, which implement device–independence, convert application drawing commands into device–specific commands for physical printers. The printer driver handles device resolution differences so that images appear the best on both 72 dpi dot–matrix printers and 300 dpi laser printers.

MACINTOSH DEVELOPMENT ENVIRONMENTS

Many different programming environments exist for the Macintosh. Since 1984 I have used Apple's Lisa Workshop and Lisa Pascal compiler to produce one commercial and many non–commercial applications. Lisa Pascal is a very powerful extension of UCSD–Pascal. As such it compiles any UCSD–Pascal program that does not contain device specific calls. Apple's Macintosh Programmer's Workshop (MPW) and MPW Pascal succeeded the Lisa Workshop. Other programming environments and languages (such as C, Forth, and Fortran) exist for the Macintosh but from my experience many of them don't provide the same professional qualities as either the Lisa Workshop or MPW. For example, LightSpeed Pascal has minimal UCSD extensions so UCSD–Pascal programs can't compile with it. The Macintosh ROM Toolbox is Pascal–oriented since the original development was done in Lisa Pascal.

INSIDE MACINTOSH

The definitive documentation for Macintosh programming is Apple's **Inside Macintosh** (IM). This book, currently a set of five volumes, describes everything a programmer needs to know about the Macintosh from how to create a window to reading data from a serial I/O port. IM contains several dozen chapters each devoted to a Toolbox manager. One chapter titled User Interface Guidelines describes in detail the user interface philosophy behind all Macintosh applications. All serious Macintosh programmers in my opinion must have this book. Besides reading IM I recommend Stephen Chernicoff's **Macintosh Revealed** and Scott Knaster's **How to Write Macintosh Software**. Chernicoff, one of IM's technical writers, presents the main concepts embodied in IM in a very readable fashion. IM requires its readers to be familiar with all the Toolbox and Operating System managers which for some programmers is very difficult reading. Knaster, a member of Apple's Macintosh Technical Support Group, provides many important techniques for more advanced programmers. If you are serious about programming the Macintosh his book is a must.

SUMMARY

Writing programs for the Macintosh is both involved and rewarding. Every Macintosh application must provide the standard user interface items such as windows, menus, and device–independent printing. The rewards appear because correctly programmed applications are easier for the user to use and require less time to learn.

<<< **The End** >>>

WDS Text I/O Unit

by William D. Smith

This is the unit which I use for reading and writing text files. I wrote it for three main reasons. First, the Pascal read and write procedures are too slow, second, I needed a way of reading files backward and third, I wanted to be able to append text to a file when there was room between the end of the file and the beginning of the next file. Since I didn't want to redo the system stuff where I didn't have too, this unit is coupled tightly with the system. The system opens the files and sets up the file information blocks (fibs). The unit then does the reading and writing, modifying the fibs as needed. Originally when I first started this unit, I tried to set the buffer size to one page so that I wouldn't have to worry about handling even and odd blocks differently. This wasn't possible because `F_LastByte`, `F_MaxByte`, `F_NxtByte` are declared as `0..512` in the `Kernel` unit and some strange results were produced when switching between what the system does and what this unit does. The systems works is done through calls to my file I/O unit, `F_Io_U`.

A text file consists of a 1K byte header and one or more 1K byte pages. One page is treated by the system as two 512 byte blocks. This means text files always have an even number of blocks. Each page, except the header, consists of lines which consist of the DLE character, the number of spaces at the beginning of the line + 32, the characters in the line followed by an end of line (EOL) character. A line (in a well defined text file) never crosses a page boundary. If the last line of the page is too long for that page, the page is filled with the zero character and the line begins on the next page.

This unit only supports reading lines. There is no way to read less then a full line. For output, you can write lines (or partial lines) to a new file or append lines to the end of a file with `P_Ln`. For input, you can read the first line, the next line, the current line, the previous line and the last line with the get procedures (prefixed by "`G_`"). Each file has a location associated with it. This location is always at the beginning of a line (or after the last line of the file). All input is referenced from this location.

The buffer, `F_Buffer` (which is part of the system fib) contains a copy of the current block (the location is in this block) of the file. This block is either being read or written. Beside interfacing to the system fib, the only real work involved in writing this unit is in handling the pages as two blocks where only one of the blocks is in the buffer at any time. Since the blocks are treated differently depending on whether it is the first or second block of a page, you have to detect this. (A line can cross the first block boundary but not the second block of a page.)

For those who have been following the discussion on MUSUS on external error detection, this unit gives examples on how I prefer to do it. The function `WriteBuf` returns `true` if everything worked and `false` otherwise. The parameter `Msg` contains the error value. When you look in `P_Ln` on how this is used, you see that in this case the message is just passed back to the caller of `P_Ln`. The alternatives being discussed were making `WriteBuf` a procedure and testing the message value after every call or having the function return the error value. Both of these methods involve more source code for determining if there is an error. Most of the time you just need to know if there is an error, not what the error is. The procedure which calls `P_Ln` is the place where you need to know what the error is so as to inform the user. This is the only place you need to decode the message. Note that the error or message talked about here is not a programming error, but a message from the system that it couldn't do a task (eg. "file not found").

```
{ WDS text input / output unit     [3.03] --- 13 Apr 88 } { |xjm$d|nx|f8|e|. }
{$Q+}
{$C (c)  William D. Smith  -- 1987 to 1990,  All rights reserved.          }
{ File:         Tx_Io_U.Text              Version 3.03    13 Apr 88
  Author:       William D. Smith.         Phone: (619) 941-4452
                P.O. Box 1139             CIS: 73007,173
                Vista, CA  92083
  Notice:      The information in this document is the exclusive
               property of William D. Smith.  All rights reserved.
               Copyright (c) 1987 to 1990.
  System:      Power System version IV.2.2
  Compiler:    Power System Pascal Compiler
  Keywords:    WDS Tx_Io_U Text Input Output Unit
  Description: This unit contains procedures for text input and output.
  Change log: (most recent first)
Date       Id   Vers  Comment
---------  ---  ----  -------
13 Apr 88  WDS  3.03  Fixed error in P_Ln.
09 Mar 88  WDS  3.01  Added G_Ln and moved AppendText to StrOps_U.
14 Feb 88  WDS  3.00  Started changes needed to use F_Io_U.
23 Oct 87  WDS  2.08  Added a writeln to CloseFile when saving.
20 Oct 87  WDS  2.07  Added Vs_Tx_Io_U and its use.
31 Aug 87  WDS  2.06  Fixed for version IV.22.
17 Aug 87  WDS  2.05  Minor syntax changes, added Caps to AppendText.
16 Jul 87  WDS  2.04  Put in version control.
19 Jun 87  WDS  2.03  Used CapStr from StrOps_U.
08 May 87  WDS  2.02  Used Glbs_U.  Removed "Tx_" prefix.
29 Apr 87  WDS  2.01  Changed P_Ln.
27 Apr 87  WDS  2.00  Derived from Text_Io (AOS).
}
{$I VERSION.TEXT}  { Declares conditional compilation flags }
unit  Tx_Io_U;
interface  {$ Tx_Io_U [3.03] 13 Apr 88 }
{ This unit only works with the Standard File System and on well defined text
  files.  Each file has a location <loc> (ie. a pointer to the begining of the
  current line) which is changed as the file is read.  In each of the functions,
  Msg is returned as M_NoError, the system ioresult, or several other errors
  depending on the function.  The value returned by the functions is true only
  if Msg = M_NoError.  OpenText and CloseText use the heap for the file
  information block.  Be careful if you use mark and release.  This unit can be
  used to write to volumes like "PRINTER:" but can not be used to read from
  volumes like "CONSOLE:" (Msg in OpenText is I_IllegalOp).
 }
uses   Glbs_U;    { WDS globals unit }
const  Vc_Tx_Iio_U  =  5;    { 09 Mar 88 }
       Vs_Tx_Io_U   = 'Tx_Io_U';
type   OpenHow      = (Tx_New,      { rewrite, file is write only }
                       Tx_Old,      { reset, file is read only }
                       Tx_Append); { append, file is write only }
       Location     = record
                        PgNum : integer;   { Page in the file }
                        ChNum : integer;   { Character number in page }
                      end { Location };
var    Vv_Tx_Io_U   : integer;
       NullLoc      : Location;  { Unused location }
  function P_Ln ( F    : FibPtr;
                  S    : Str_255;
                  N    : integer;
              var Msg : integer) : boolean;
  { Put line.  Writes the string S and N end of lines to the file F.  N = 0 is
    allowed.  Msg is M_NoError or ioresult.
   }
```

```
function G_FstLn ( F    : FibPtr;
                  var S    : Str_255;
                  var Msg : integer) : boolean;
{ Get first line.  This function reads the first line from the file F into S.
  Msg can be M_NoError, ioresult, or M_Empty (file is empty).  <loc> is
  immediately past the line just read.  This function sets the file up to be
  read in the forward direction. (ie.  After G_FstLn, G_NxtLn returns the
  second line.  After G_FstLn, G_PrvLn returns the same line and and leaves
  <loc> at the begining of file.)
}
function G_NxtLn ( F    : FibPtr;
                  var S    : Str_255;
                  var Msg : integer) : boolean;
{ Get next line.  This function reads the next line from the file F into S.
  Msg can be M_NoError, ioresult, or M_Eof (at end of file).  <loc> is
  immediately past the line just read.
}
function G_CurLn ( F    : FibPtr;
                  var S    : Str_255;
                  var Msg : integer) : boolean;
{ Get current line.  This function reads the current line from the file F into
  S.  Msg can be M_NoError, ioresult, or M_Eof (at end of file).  <loc> is not
  changed.
}
function G_PrvLn ( F    : FibPtr;
                  var S    : Str_255;
                  var Msg : integer) : boolean;
{ Get previous line.  This function reads the previous line from the file F
  into S.  Msg can be M_NoError, ioresult, or M_Bof (at begining of file).
  <loc> is at the begining of the line just read.
}
function G_LstLn ( F    : FibPtr;
                  var S    : Str_255;
                  var Msg : integer) : boolean;
{ Get last line.  This function reads the last line from the file F into S.
  Msg can be M_NoError, ioresult, or M_Empty (file is empty).  <loc> is at the
  begining of the line just read.  This function sets the file up to be read
  in the reverse direction. (ie.  After G_LstLn, G_PrvLn returns the second to
  last line.  After G_LstLn, G_NxtLn returns the same line and leaves <loc> at
  the end of file.)
}
function G_Ln (  Dir  : Direction;
                   F    : FibPtr;
              var S    : Str_255;
              var Msg : integer) : boolean;
{ Get line.  This function calls one of the five previous "get" functions
  depending on the value of Dir.
}
function CmpLoc (Src : Location;  Tgt : Location) : CmpType;
{ Compare locations.  This function compares the location Src and Tgt and
  returns less then (Lt), equal (Eq), or greater then (Gt).
}
function GetLoc ( Dir  : Direction;
                   F    : FibPtr;
               var Loc  : Location;
               var Msg  : integer) : boolean;
{ Get location.  This procedure gets the location in the file F asked for by
  Dir.  Only F_What, C_What, and L_What are allowed for Dir.  This procedure
```

```
    is only valid for files opened as Tx_Old.  Loc is returned as NullLoc in all
    invalid or error situations.  Msg is M_NoError or ioresult 3 (bad mode).
  }
  function SetLoc ( F     : FibPtr;
                  var Loc  : Location;
                  var Msg  : integer) : boolean;
  { Set location.  This procedure sets the current location in the file F.  If
    the location is not at the begining of a line it is forced to the begining
    of the the line.  This procedure is only valid for files opened as Tx_Old.
    Msg is M_NoError or ioresult.
  }
  function OpenText (var   F     : FibPtr;
                           Name  : Str_23;
                           How   : OpenHow;
                       var Msg   : integer) : boolean;
  { Open text file.  This function opens the file called Name.  F is returned as
    a pointer to the file information block (FIB) if the file is opened and as
    Closed if the file was not opened.  How determines how the file will be
    opened.  Msg can be M_NoError, ioresult, M_NoHeap (no room to allocate the
    FIB).  <loc> is at the begining of the first line.
  }
  procedure CloseText (var F : FibPtr;  Save : boolean);
  { Close text file.  The file F is closed and the FIB is disposed.  F is
    returned as Closed.  Save and how the file was opened determine how the file
    is to be closed.

          How       Save      Closed    Comment
          ------    -----     ------    -------
          New       true      lock      New file made permanent
          New       false     purge     No file
          Old       true      normal    File unchanged
          Old       false     normal    File unchanged
          Append    true      lock      File increased in size
          Append    false     crunch    File unchanged
  }
implementation
uses  Kernel (F_BlkSize, F_TypeText, IoRsltWd, FibP, Fib),
      F_Io_U;       { WDS file I/O unit }
const Nul          = 0;
      Eo           = 13;
      Dle          = 16;
      Sto          = 196;
      StrSize      = 255;   { Maximum size of input/output strings }
      BufSize      = 512;   { BufSize := F_Io_U .PageSize * 2; }
      MaxBuf       = 511;   { MaxBuf := BufSize - 1; }
      BlksInPg     = 1;
      BlkSize      = 512;   { Bytes in a block }
  function Ck_File (F : FibPtr;  Ro : boolean;
                    var Fp : FibP;  var Msg : integer) : boolean;
  begin
    if F = Closed then  Msg := ord (I_NotOpen)
    else
      begin
        pmachine (^Fp, (F), Sto);  { Fp := F; }
        if Fp^ .F_ReadOnly = Ro then  Msg := M_NoError
        else  Msg := ord (I_BadMode);
      end { else };
    Ck_File := Msg = M_NoError;
  end { Ck_File };

  function ReadBuf (Fp : FibP;  Back : boolean;
                    var Msg : integer) : boolean;
  { If Back read F_NxtBlk - 2 else read F_NxtBlk.  Set F_NxtByte and F_NxtBlk. }
  var Done : boolean;  F : FibPtr;
  begin
    Msg := M_NoError;
    Done := false;
    pmachine (^F, (Fp), Sto);  { F := Fp; }

    with Fp^ do begin
      repeat
        if Back then
          if F_NxtBlk <= 3 then  Msg := M_Bof
          else
        else if F_NxtBlk >= F_MaxBlk then  Msg := M_Eof;

        if Msg = M_NoError then
          begin
            if Back then  F_NxtBlk := F_NxtBlk - 2;

            F_NxtBlk := F_NxtBlk +
                        BlockIo (F, F_Buffer, BlksInPg, F_NxtBlk, true);
            Msg := ioresult;

            if Msg = M_NoError then
              if F_Buffer [0] <> chr (Nul) then  { Buf not empty }
                begin
                  Done := true;

                  if Back then
                    F_NxtByte := BufSize + scan (- BufSize, <> chr (Nul),
                                                 F_Buffer [MaxBuf])
                  else  F_NxtByte := 0;
                end { if if };
          end { if };
      until Done or (Msg <> M_NoError);
    end { with };

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

  function WriteBuf (Fp : FibP;  var Msg : integer) : boolean;
  { Write F_NxtBlk.  Set F_NxtByte and F_NxtBlk. }
  var F : FibPtr;
  begin
    pmachine (^F, (Fp), Sto);  { F := Fp; }

    with Fp^ do begin
      if F_NxtByte < BufSize then
        fillchar (F_Buffer [F_NxtByte], BufSize - F_NxtByte, Nul);

      F_NxtBlk := F_NxtBlk + BlockIo (F, F_Buffer, BlksInPg, F_NxtBlk, false);
      F_NxtByte := 0;
    end { with };

    Msg := ioresult;
    WriteBuf := Msg = M_NoError;
  end { WriteBuf };

  function P_Ln { (F : FibPtr;  S : Str_255;  N : integer;
                  var Msg : integer) : boolean };
  label  1, 2;
```

```
var    Fp : FibP;   I, J : integer;   NeedDle : boolean;
begin
  if not Ck_File (F, false, Fp, Msg) then  goto 1;

  with Fp^ do begin
    if not F_SoftBuf then  { CONSOLE:, PRINTER:, etc. }
      begin
        unitwrite (F_Unit, S [1], length (S));

        if N > 0 then
          begin
            if N >= 256 then
              begin
                fillchar (S [0], 256, chr (Eol));

                repeat
                  unitwrite (F_Unit, S [0], 256);
                  N := N - 256;
                until N <= 255;
              end { if }
            else  fillchar (S [1], N, chr (Eol));

            if N > 0 then  unitwrite (F_Unit, S [1], N);
          end { if };
      end { if }
    else { if F_SoftBuf then }
      begin
        if F_NxtByte = 0 then  NeedDle := true
        else if F_Buffer [F_NxtByte - 1] = chr (Eol) then  NeedDle := true
        else  NeedDle := false;

        if N > 0 then  { remove spaces from end of string }
          S [0] := chr (length (S) +
                    scan (- length (S), <> ' ', S [length (S)]));

        if NeedDle then
          begin
            I := 1 + scan (length (S), <> ' ', S [1]);
            J := 2;
          end { if }
        else
          begin
            I := 1;  J := 0;

            if (length (S) = 0) and (N > 0) then  { Back up F_NxtByte }
              F_NxtByte := F_NxtByte + scan (- F_NxtByte, <> ' ',
                                                F_Buffer [F_NxtByte - 1]);
          end { else };

        if N > 0 then  J := J + 1;

        J := J + (length (S) - I) + 1;

        if J <= BufSize - F_NxtByte then  goto 2  { fits }
        else if not odd (F_NxtBlk) then  { split line }
          begin
            if NeedDle then
              begin
                if F_NxtByte > MaxBuf then
                  if not WriteBuf (Fp, Msg) then  goto 1;

                F_Buffer [F_NxtByte] := chr (Dle);
                F_NxtByte := F_NxtByte + 1;

                if F_NxtByte > MaxBuf then
                  if not WriteBuf (Fp, Msg) then  goto 1;
                F_Buffer [F_NxtByte] := chr (31 + I);
                F_NxtByte := F_NxtByte + 1;
              end { if };
            if F_NxtByte > MaxBuf then
              if not WriteBuf (Fp, Msg) then  goto 1;
            J := (length (S) - I) + 1;
            if J + F_NxtByte > BufSize then
              J := BufSize - F_NxtByte;
            moveleft (S [I], F_Buffer [F_NxtByte], J);
            F_NxtByte := F_NxtByte + J;
            if I + J <= length (S) then
              begin
                I := I + J;
                J := (length (S) - I) + 1;
                if F_NxtByte > MaxBuf then
                  if not WriteBuf (Fp, Msg) then  goto 1;
                moveleft (S [I], F_Buffer [F_NxtByte], J);
                F_NxtByte := F_NxtByte + J;
              end { if };
            if N > 0 then
              begin
                if F_NxtByte > MaxBuf then
                  if not WriteBuf (Fp, Msg) then  goto 1;
                F_Buffer [F_NxtByte] := chr (Eol);
                F_NxtByte := F_NxtByte + 1;
                N := N - 1;
              end { if };
          end { else if }
        else  { all in following block }
          begin
            if not WriteBuf (Fp, Msg) then  goto 1;
          2:
            if NeedDle then
              begin
                F_Buffer [F_NxtByte] := chr (Dle);
                F_NxtByte := F_NxtByte + 1;
                F_Buffer [F_NxtByte] := chr (31 + I);
                F_NxtByte := F_NxtByte + 1;
              end { if };
            moveleft (S [I], F_Buffer [F_NxtByte], (length (S) - I) + 1);
            F_NxtByte := F_NxtByte + (length (S) - I) + 1;
            if N > 0 then
              begin
                F_Buffer [F_NxtByte] := chr (Eol);
                F_NxtByte := F_NxtByte + 1;
                N := N - 1;
              end { if };
          end { else };
        for I := 1 to N do begin
          if not P_Ln (F, '', 1, Msg) then  goto 1;
        end { for };
      end { else };
  end { with };
1:
  P_Ln := Msg = M_NoError;
end { P_Ln };

function G_FstLn { (F : FibPtr;  var S : Str_255;
                    var Msg : integer) : boolean };
var Fp : FibP;
begin
  S [0] := chr (0);   { S := ''; }

  if Ck_File (F, true, Fp, Msg) then
    begin
      with Fp^ do begin
        if F_NxtBlk = 3 then   { block 2 already in buffer }
          F_NxtByte := 0
        else
          begin
            F_NxtBlk := 2;   { first text block }
            F_NxtByte := BufSize;   { forces ReadBuf by G_NxtLn }
          end { else };
      end { with };

      if G_NxtLn (F, S, Msg) then ;
    end { if };

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

function G_NxtLn { (F : FibPtr;  var S : Str_255;
                    var Msg : integer) : boolean };
label  1;
var    Fp : FibP;  I, J, Si : integer;

  function G_Eol : boolean;
  begin
    with Fp^ do begin
      I := F_NxtByte + scan (BufSize - F_NxtByte, = chr (Eol),
                             F_Buffer [F_NxtByte]);
      if I = BufSize then
        if F_Buffer [I - 1] = chr (Nul) then   { scan backward }
          I := I + scan (- (I - F_NxtByte), <> chr (Nul), F_Buffer [I - 1]);

      J := I - F_NxtByte;

      if Si + J > StrSize then
        Msg := ord (I_BufOvfl)
      else
        begin
          moveleft (F_Buffer [F_NxtByte], S [Si + 1], J);
          Si := Si + J;
        end { else };
    end { with };

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

begin { G_NxtLn }
  S [0] := chr (0);
  if not Ck_File (F, true, Fp, Msg) then  goto 1;

  with Fp^ do begin
    if F_NxtByte > MaxBuf then
      if not ReadBuf (Fp, false, Msg) then  goto 1;

    if F_Buffer [F_NxtByte] = chr (Nul) then   { no more text in buffer }
      if not ReadBuf (Fp, false, Msg) then  goto 1;
      if F_Buffer [F_NxtByte] = chr (Dle) then
        begin
          F_NxtByte := F_NxtByte + 1;

          if F_NxtByte > MaxBuf then
            if not ReadBuf (Fp, false, Msg) then  goto 1;

          Si := ord (F_Buffer [F_NxtByte]) - 32;
          fillchar (S [1], Si, ' ');

          F_NxtByte := F_NxtByte + 1;

          if F_NxtByte > MaxBuf then
            if not ReadBuf (Fp, false, Msg) then  goto 1;
        end { if }
      else  Si := 0;

      if not G_Eol then  goto 1;

      if I > MaxBuf then
        begin
          if ReadBuf (Fp, false, Msg) then
            begin
              if not G_Eol then  goto 1;
            end { if }
          else if Msg = M_Eof then
            Msg := M_NoError
          else  goto 1;
        end { if };

      F_NxtByte := I + 1;
      S [0] := chr (Si);
    end { with };
  1:
    G_NxtLn := Msg = M_NoError;
  end { G_NxtLn };

  function G_CurLn { ( F : FibPtr;  var S : Str_255;
                       var Msg : integer) : boolean };
  var Loc : Location;
  begin
    S [0] := chr (0);  { S := ''; }

    if GetLoc (C_What, F, Loc, Msg) then
      if G_NxtLn (F, S, Msg) then
        if SetLoc (F, Loc, Msg) then ;

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

  function G_PrvLn { (F : FibPtr;  var S : Str_255;
                       var Msg : integer) : boolean };
  label  1;
  var    Fp : FibP;  I, J, K, Si : integer;
  begin
    S [0] := chr (0);  { S:= ''; }

    if not Ck_File (F, true, Fp, Msg) then  goto 1;

    with Fp^ do begin
      if F_NxtByte = 0 then
        if not ReadBuf (Fp, true, Msg) then  goto 1;

      I := F_NxtByte - 1;

      if F_Buffer [I] = chr (Eol) then  { skip Eol }
        begin
          F_NxtByte := I;  I := I - 1;
      if I < 0 then
        if not ReadBuf (Fp, true, Msg) then  goto 1
        else  I := F_NxtByte - 1;
    end { if };

  I := I + 1 + scan (- F_NxtByte, = chr (Eol), F_Buffer [I]);

  if (I > 0) or ((I = 0) and (F_Buffer [0] = chr (Dle))) then  { Eol }
    begin
      if F_Buffer [I] = chr (Dle) then
        begin
          Si := ord (F_Buffer [I + 1]) - 32;
          fillchar (S [1], Si, ' ');
          J := I + 2;
        end { if }
      else  begin  J := I;  Si := 0;  end { else };

      K := F_NxtByte - J;

      if Si + K > StrSize then
        begin
          Msg := ord (I_BufOvfl);
          goto 1;
        end { if };

      moveleft (F_Buffer [J], S [Si + 1], K);
      S [0] := chr (Si + K);
      F_NxtByte := I;
    end { if }
  else
    begin
      Si := StrSize - F_NxtByte + 1;

      if Si <= 0 then
        begin
          Msg := ord (I_BufOvfl);
          goto 1;
        end { if };

      moveleft (F_Buffer [0], S [Si], F_NxtByte - I);
      K := StrSize - Si + 1;

      if ReadBuf (Fp, true, Msg) then
        begin
          I := F_NxtByte + scan (- F_NxtByte, = chr (Eol),
                                 F_Buffer [F_NxtByte - 1]);

          if F_Buffer [I] = chr (Dle) then
            begin
              if I + 1 = F_NxtByte then  { expansion in string }
                begin
                  J := ord (S [Si]) - 32;
                  Si := Si + 1;
                  K := K - 1;

                  if J + K > StrSize then
                    begin
                      Msg := ord (I_BufOvfl);
                      goto 1;
                    end { if };

                  fillchar (S [1], J, ' ');
                  moveleft (S [Si], S [J + 1], K);
                  S [0] := chr (J + K);
                end { if }
              else
```

```
                    begin
                      J := ord (F_Buffer [I + 1]) - 32;

                      if J + F_NxtByte - (I + 2) + K > StrSize then
                        begin
                          Msg := ord (I_BufOvfl);
                          goto 1;
                        end { if };

                      fillchar (S [1], J, ' ');
                      moveleft (F_Buffer [I + 2], S [J + 1],
                                 F_NxtByte - (I + 2));
                      moveleft (S [Si], S [J + F_NxtByte - (I + 2) + 1], K);
                      S [0] := chr (J + F_NxtByte - (I + 2) + K);
                    end { else };
                end { if }
              else if K + F_NxtByte - I > StrSize then
                begin
                  Msg := ord (I_BufOvfl);
                  goto 1;
                end { else if }
              else
                begin
                  moveleft (F_Buffer [I], S [1], F_NxtByte - I);
                  moveleft (S [Si], S [F_NxtByte - I + 1], K);
                  S [0] := chr (F_NxtByte - I + K);
                end { else };

              F_NxtByte := I;
            end { if }
          else if Msg = M_Bof then
            begin
              moveleft (S [Si], S [1], K);
              S [0] := chr (K);
              Msg := M_NoError;
            end { else if };
        end { else };
    end { with };
1:
    G_PrvLn := Msg = M_NoError;
  end { G_PrvLn };

  function G_LstLn { (F : FibPtr;  var S : Str_255;
                      var Msg : integer) : boolean };
  var Fp : FibP;
  begin
    S [0] := chr (0);  { S := ''; }

    if Ck_File (F, true, Fp, Msg) then
      begin
        with Fp^ do begin
          if F_NxtBlk >= F_MaxBlk then  { last block already in buffer }
            F_NxtByte := BufSize
          else
            begin
              F_NxtBlk := F_MaxBlk + 1;  { last block }
              F_NxtByte := 0;  { forces ReadBuf by G_PrvLn }
            end { else };
        end { with };

        if G_PrvLn (F, S, Msg) then ;
      end { if };
  G_LstLn := Msg = M_NoError;
end { G_LstLn };

function G_Ln { (Dir : Direction;  F : FibPtr;
                 var S : Str_255;  var Msg : integer) : boolean };
begin
  case Dir of
    F_What : G_Ln := G_FstLn (F, S, Msg);
    P_What : G_Ln := G_PrvLn (F, S, Msg);
    C_What : G_Ln := G_CurLn (F, S, Msg);
    N_What : G_Ln := G_NxtLn (F, S, Msg);
    L_What : G_Ln := G_LstLn (F, S, Msg);
  end { cases };
end { G_Ln };

function CmpLoc { (Src : Location;  Tgt : Location) : CmpType };
begin
  if Src = Tgt then  CmpLoc := Eq
  else if Src .PgNum > Tgt .PgNum then  CmpLoc := Gt
  else if Src .PgNum < Tgt .PgNum then  CmpLoc := Lt
  else { if Src .PgNum = Tgt .PgNum then }
    if Src .ChNum > Tgt .ChNum then  CmpLoc := Gt
    else if Src .ChNum < Tgt .ChNum then  CmpLoc := Lt
    else  CmpLoc := Eq;
end { CmpLoc };

function GetLoc { (Dir : Direction;  F : FibPtr;
                    var Loc : Location;  var Msg : integer) : boolean };
var Fp : FibP;
begin
  Loc := NullLoc;

  if Ck_File (F, true, Fp, Msg) then
    with Fp^, Loc do begin
      case Dir of
        F_What : begin  PgNum := 1;  ChNum := 0;  end { case F_What };

        C_What : begin
                   PgNum := (F_NxtBlk - 1) div 2;
                   ChNum := F_NxtByte;

                   if not odd (F_NxtBlk) then
                     ChNum := ChNum + BlkSize;
                 end { case C_What };

        L_What : begin
                   PgNum := (F_MaxBlk - 1) div 2;
                   ChNum := BufSize + BufSize;
                 end { case L_What };

        P_What,
        N_What : Msg := ord (I_IllegalOp);
      end { cases };
    end { with };

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

function SetLoc { (F : FibPtr;  var Loc : Location;
                    Msg : integer) : boolean };
var Fp : FibP;  I : integer;
begin
  if Ck_File (F, true, Fp, Msg) then
    with Fp^, Loc do begin
      I := PgNum * 2;
```

```
        if ChNum >= BlkSize then  I := I + 1;

        if (I < 2) or (I > F_MaxBlk) then
          Msg := ord (I_BadBlock)
        else
          begin
            if (I <> F_NxtBlk - 1) or (F_NxtBlk >= F_MaxBlk) then
              begin  { need to read block }
                F_NxtBlk := I;
                SetLoc := ReadBuf (Fp, false, Msg);
              end { if }
            else  Msg := M_NoError;

            if Msg = M_NoError then
              begin
                if ChNum >= BlkSize then  F_NxtByte := ChNum - BlkSize
                else  F_NxtByte := ChNum;

                if F_NxtByte > 0 then
                  F_NxtByte := F_NxtByte + scan (- F_NxtByte, = chr (Eol),
                                                 F_Buffer [F_NxtByte - 1]);
              end { if }
            else if Msg = M_Eof then  F_NxtByte := 0
            else  F_NxtByte := BufSize;
          end { else };
      end { with };

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

  function OpenText { (var F : FibPtr;  Name : Str_23;  How : OpenHow;
                       var Msg : integer) : boolean };
  var Fp : FibP;  Old : boolean;  S : Str_255;
  begin
    Msg := M_NoError;
    Old := How <> Tx_New;

    if OpenFile (F, Name, TxtFile, Old, Msg) then
      begin
        pmachine (^Fp, (F), Sto);  { Fp := F; }

        with Fp^ do begin
          if F_SoftBuf then
            begin
              if F_Type <> F_TypeText then
                begin
                  CloseFile (F, false);
                  Msg := ord (I_BadFileType);
                end { if }
              else if How = Tx_Append then
                begin
                  F_Readonly := true;

                  if G_LstLn (F, S, Msg) then
                    if G_NxtLn (F, S, Msg) then ;

                  F_Readonly := false;
                end { else if }
              else
                begin
                  F_Readonly := How = Tx_Old;
                  F_NxtByte := 0;  { undo stuff from open }
                end { else };
            end { else if }
          else if How = Tx_Old then  { CONSOLE: input not allowed }
```

```
              begin
                CloseFile (F, false);
                Msg := ord (I_IllegalOp);
              end { else if }
            else if F_IsBlkd then  { Blocked volume I/O not allowed }
              begin
                CloseFile (F, false);
                Msg := ord (I_BadFileType);
              end { else if }
            else  { PRINTER:, CONSOLE: output, ?? don't need buffer in this çase }
              F_Readonly := false;
          end { with };
        end { if };

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

    procedure CloseText { (var F : FibPtr;  Save : boolean) };
    var Fp : FibP;  Msg : integer;
    begin
      if F <> Closed then
        begin
          pmachine (^Fp, (F), Sto);  { Fp := F; }
          with Fp^ do begin
            if Save and not F_Readonly and F_SoftBuf then
              begin
                F_NxtByte := F_NxtByte + scan (- F_NxtByte, <> ' ',
                                              F_Buffer [F_NxtByte - 1]);
                if F_NxtByte > 0 then
                  if F_Buffer [F_NxtByte - 1] <> chr (Eol) then
                    begin
                      if F_NxtByte > MaxBuf then
                        if WriteBuf (Fp, Msg) then ;
                      F_Buffer [F_NxtByte] := chr (Eol);
                      F_NxtByte := F_NxtByte + 1;
                    end { if }
                  else
                else if odd (F_NxtBlk) then
                  begin
                    F_Buffer [F_NxtByte] := chr (Eol);
                    F_NxtByte := F_NxtByte + 1;
                  end { else if };
                F_BufChngd := F_NxtByte > 0;
                F_Modified := F_BufChngd or (F_NxtBlk > 2);
                if F_Modified then  F_NxtBlk := F_NxtBlk + 1
                else  Save := false;
              end { if };
          end { with };
          CloseFile (F, Save);
        end { if };
    end { CloseText };

  begin
    Vv_Tx_Io_U := Vc_Tx_Io_U;
    Ck_Version (Vv_Glbs_U, Vc_Glbs_U, Vs_Tx_Io_U, Vs_Glbs_U);
    Ck_Version (Vv_F_Io_U, Vc_F_Io_U, Vs_Tx_Io_U, Vs_F_Io_U);
    fillchar (NullLoc, sizeof (NullLoc), 0);
    *** ;
  end {$Q- Tx_Io_U }.
```

From The Editor

by Tom Cattrall

This issue has taken longer to get out than I'd like because I decided to go for 36 pages rather than 24 and it took extra effort to collect enough material to fill it out. The Text I/O Unit article by William Smith is 14 pages and other material already amounted to more than 10 pages so it wouldn't all fit in 24 pages and 36 is the next increment.

I'm still getting the process worked out for producing the newsletter. Material comes from various sources, sometimes voluntarily, sometimes because I see something and ask permission to use it. Mail from the USUS P.O. Box is sent on by William Smith. The treasurer's reports arrive via electronic mail from Bob Clark. The meeting minutes written up by Keith Frederick also arrive via electronic mail. The letter and review by James Harding were the first items that I've received as a result of the first issue. I hope others follow his lead in sending in any tidbits they have that readers will find interesting.

Once enough has been collected and laid out, I send the pages off to Robert Geeslin in Oklahoma. He takes them to a local printer and then about a week later picks up the printed newsletters. In the meantime, I notify Felix Bearden to run off address labels and send them to Robert. Robert merges newsletters, labels, and stamps and then takes it all to the post office. And then a month later the process starts again.

The last issue was delayed because I either hadn't been told about the step of notifying Felix to send labels, or probably more likely, had forgot about that step.

Previous issues of the newsletter had arrived here looking like a truck had left tire tracks across the front and back pages. I asked Robert if he had seen this but as his copy is handled locally, it didn't have this. He took photocopies of the pages that I sent him and discussed it with the local postmaster. I didn't hold out a lot of hope that anything could be done but somewhat to my surprise, my copy arrived unscathed. So, congratulations to Robert for being so persuasive with the post office.

File Archiving Project

I didn't get enough time to do anything with the file archiving project but hope to do so for the next issue. My current thinking is that rather than develop something entirely new, we'd be better off to adopt one of the existing archive structures. That way, we could exchange material with other users that are already using the archive scheme. From a technical and availability standpoint, the zoo archive utility is my first choice. It can handle P-system style file names and is available on a wide variety of systems.

If you were to count up the megabytes of information stored in each archive format, the venerable arc utility would be the easy winner. For that reason, it would be nice to have an implementation of arc that runs on the P-system even though it won't handle the file names. The arc format is also quite a bit more complex to implement than is zoo's.

I have source code in C for both zoo and arc, and have ported the compression algorithm used in zoo to Pascal. So, I'd like to get a P-system zoo archiver going as the first step. Then if enough resources (volunteers) still exist, it would be nice to go on and implement something to handle arc format.

Board Meetings

Be sure to read the note from Alex Kleider on Page 6. There wasn't a quorum at the March 13'th meeting either (Henry and I were the only board members) so we quickly adjourned to try again April 11th. Will try using post card reminders to make sure everyone is aware of the new meeting time: 2nd Wednesday of month at 6:30 PM Pacific time.

If we can ever get a quorum I'd like to suggest that the board meetings be held less often. Perhaps every 2 or 3 months. And when meetings are held, give advance notice by postcard giving date, time, and agenda. I don't think that more frequent meetings are needed now and by holding fewer meetings but making a big deal when we do, perhaps everyone will find it easier to remember to attend.

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 MSDOS 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 <u>underline</u>.

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 "?? ??".

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.

TULSA, OK 741 APR 7 PM 1990

NewsLetter Publication Dates

Issue	Due Date For All Newsletter Material
May / Jun 1990	May 4, 1990
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]

TULSA APR PM 19