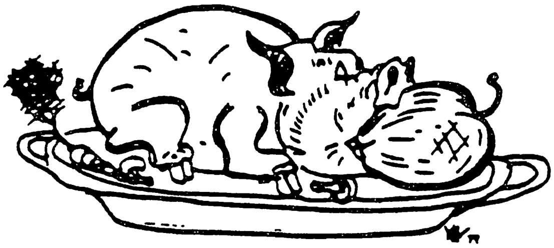
H.A.A.U.G.



HOUSTON AREA APPLE USERS GROUP

# THE APPLE BARREL

VOLUME 5 NO. 8 OCTOBER/NOVEMBER 1982

PRESIDENT, MIKE KRAMER VICE PRES., BRIAN WHALEY
EDITOR, MIKE KRAMER

*** CONTENTS ***

# HAAUG APPLE BARREL

2218 Running Springs
Kingwood, TX 77339

## CLUB NOTES

The HOUSTON AREA APPLE USERS GROUP is an Apple user club, not affiliated with Apple, Inc., or any retail computer store. HAAUG is a member of the International Apple Core and supports its publications and purposes. General membership meetings are held on the second Thursday of each month in the rear chapel of Memorial Lutheran Church, 5800 Westheimer, between Chimney Rock and Jungman Library, beginning at 6:30 P.M. An additional general meeting is held beginning at noon the third Saturday of each month at the UT School of Public Health in the Medical Center at 6905 Bertner at Holcomb. This meeting features tutorials, special interest group meetings, problem-solving sessions, and access to the HAAUG software library. The meeting is held in the main floor meeting room to the left of the entrance. Bring your Apples!!

---==*==---

## OFFICERS / EXECUTIVE BOARD

| | |
|---|---|
| President | Mike Kramer |
| Vice Pres | Brian Whaley |
| Treasurer | Wally Edmiston |
| Secretary | Ruth Dill |
| Software Lib. | Jim Good |
| Hardcopy Lib. | Larry Baumann |
| Membership | Lee Gilbreath |
| IAC Rep. | Vacant |

## APPLE HOTLINE
?????????

The APPLE HOTLINE has been established to provide an easy means to learn of meeting topics, news, etc. It can also be used to obtain answers to puzzling Apple - related questions. If you get a recording, leave your name, date, and time. You should get a return call within 24 hours.

---===*===---

## MEMBERSHIP INFORMATION

New memberships are $30 and include the HAAUG starter kit. Renewals are $20 per year. Make checks payable to Houston Area Apple Users Group and mail to Lee Gilbreath, 3609 Glenmeadow, Rosenberg,TX 77471.Do not send renewals to the Apple Barrel.

---==*==---

## APPLE BARREL REPRINT POLICY

Unless otherwise noted within the program or article, any original material published herein may be reprinted without permission by any non-profit Apple club, group, or newsletter, provided proper credit is given to the Apple Barrel and the author.

## ADVERTISING RATES

| FULL PAGE | HALF PAGE | QTR PAGE | 6TH PAGE | 8th PAGE |
|---|---|---|---|---|
| $50 | $28 | $18 | $15 | $10 |

Advertisements should be submitted in camera ready form to H.A.A.U.G., 2218 Running Springs, Kingwood, TX, 77339, by the 1ST of the month. Charges will be billed and accompanied by a copy of the Apple Barrel containing the ad.

## THOUGHTS FROM THE PRES

Following the last Thursday meeting a meeting was held to organize a nominating committee to select a slate of officers for for next year. The committee is being chaired by Robin Cox, who has the benefit of having served on the committee last year. By now most of you know that I do not plan to run for another term. I feel that with a year as Vice President and another as President I should move over and let others set the course that HAAUG should take. I've had good times in office and have experienced a good bit of frustration at not having the time to provide the leadership the club needs. I do feel on the other hand that some things got better, such as the Apple Barrel and the format of the Saturday meeting. I plan to continue to be active in the running of HAAUG, both as editor of the Apple Barrel (unless someone else wants to do it) and as co-chairman of the Apple /// Special Interest Group.

The new officers face the challenge of handling the growth which is inevitable. I'm not sure how many Apples are sold each month in Houston, but with 20+ stores selling systems at mail order prices, we must be seeing fewer than 10% joining HAAUG. With the publicity we will gain at Applefest, we could instantly increase our rolls by 50%. How are we going to organize to handle the growth??? Cast your votes wisely because the new officers will be faced with this question and it won't be easy.

I would like to express my appreciation for your support and the help given by the other officers, special interest group chairmen, and others who have given so generously of their time. At the risk of leaving anyone out, I'd like to acknowledge the contributions made by Brian Whaley who has covered for me when I was travelling and consistently provided good speakers, Ruth Dill who was willing to do more secretarial tasks than she was actually given, Wally Edmiston who tried to keep me from bankrupting HAAUG with Apple Barrel printing costs, Jim Good who somehow keeps his sanity while trying to satisy new members hunger for software, Lee Gilbreath who spends more than his share of time as membership chairman, DeWayne Van Hoozer who does a little bit of everything, Dr.Carl Hacker who has made it possible for HAAUG to use the UT facility,the list could go on and on...

Many of you probably noticed the lack of a Hotline number on the inside front cover and that the position of IAC Representative is vacant. What you probably do not realize is that by the time this is received, DeWayne Van Hoozer, past President of HAAUG, IAC representative, and answerer of the HAAUG Hotline will have moved to Chatsworth, California, where he will work for Datamost, a well-known software publisher (Snack Attack, Genasys). DeWayne will be missed, but, based on past peformance, he will return. Until the new hotline number is known, call me at 358-6687. When we have a new hotline, please forget mine!

LIBINFO

## The UCSD Pascal Language System: Libraries, Segments and Overlays

by Ben P. Baston

Southwest
Computer Consultants,
Inc.

The UCSD Pascal Language System is an entire operating system with utility programs and upper-level language(s). It is available for almost any computer. Programs developed on one computer can be directly usable on any other UCSD-equipped computer (except for machine-language routines).

The UCSD Language System allows programmers to provide often-used routines and procedures for use in any program without repeating the routines' code or compilation. This is common among main-frame computer operating systems and is known as the library facility.

The Language System also supports program segmentation and overlaying, which together allow very large programs to run on small-memory computers. Segmentation and overlaying are important features needed for developing large and sophisticated applications on small computers.

Though the information presented is generally true of the UCSD Language System, most details are specific to Apple Pascal Version 1.1, including all FORTRAN details.

### Introduction

Information is entered into the UCSD system by using the utility program named SYSTEM.EDITOR. The Editor allows the user to create and revise text files. If the text file represents a program or program part, it is a source file that another utility program (SYSTEM.COMPILER or SYSTEM.ASSMBLER) can attempt to translate into a code file. The translation process is called "compiling", though the term assembling can be used when the translation is into a code directly readable by the computer. Only a code file can be used by the system.

To produce a finished program, it may be necessary to take several program parts and combine them into one executable code file. The process of joining separately-compiled program segments into a single finished program is called "linking". This procedure is handled by the utility program named SYSTEM.LINKER.

Apple II FORTRAN users should note that there is an extra step in properly linking FORTRAN code files. Due to a bug in its compiler, every FORTRAN code file must have its linking information corrected by a program named FORTFIX before using the Linker. By the way, FORTRAN and Pascal code files are identical, both representing "pseudo" machine code which is interpreted at run time into native machine code.

LIBINFO

With both Pascal and FORTRAN, the programmer may group related subroutines or functions together and compile them even though they do not represent a program. Such a compilation code file is called a "UNIT". In FORTRAN, a UNIT must begin with either SUBROUTINE or [type] FUNCTION. A Pascal UNIT begins with the word UNIT and is divided into two parts by the words INTERFACE and IMPLEMENTATION. As demonstrated below, programming in segments is both a powerful and a simple tool. It is the key to unlocking the potential of the UCSD Pascal Operating System.

Library Files

With the Language System's library facility, related routines are grouped together in a compiled UNIT. The UNIT is not a complete program. It is a potential program part or segment. Several of these UNITs can be put in one file - called a Library File - by using the appropriately named program "LIBRARY".

Apple II FORTRAN users MUST replace the program LIBRARY with an updated version supplied with the FORTRAN compiler in order to insure proper Library File behavior. The new version should be given the old name, LIBRARY.CODE.

| | NAME | LENGTH | DATE |
|---|---|---|---|
| new version: | FORTLIB.CODE | 9 blocks | 19-Jan-80 |
| old version: | LIBRARY.CODE | 8 blocks | 15-Sep-80 |

The term "slot" is a convenient way to reference a UNIT after it has been installed in a Library File. A Library File has a capacity of 16 UNITs, each of which is a potential program segment. Library slots are numbered from 0 through 15. In the discussion that follows, do not confuse a Library File slot number with UCSD Operating System Program Segments, which are numbered from 0 to 31.

Two types of UNITs are found in Version 1.1 Library Files (Version IV and later use 100% Intrinsic UNITs):

1. Regular UNIT - This type must be linked into each program which uses it. Its code is literally duplicated into the using segment, so the Regular UNIT itself is not needed on-line after linking. Its main advantage is giving the capability to compile a large program in smaller pieces, to speed development time or to compile a program otherwise too large to be compiled all in one piece. A Regular UNIT requires manual linking unless it is contained in a special file called SYSTEM.LIBRARY on the boot volume. In that case, linking is required but can be "automated".

2. Intrinsic UNIT - As its name implies, these UNITS become an extension of the language, adding whole groups of commands and capabilities accessible by all programs. An Intrinsic UNIT's code is never duplicated, just accessed and used by the program needing it. Intrinsic UNITs may use other Intrinsic UNITS and may contain linked External Segments (explained below), but not linked Regular UNITs. An Intrinsic UNIT never requires linking. All programs that use it benefit from Intrinsic UNIT

LIBINFO

improvements without further action, so long as improvements are made only in the IMPLEMENTATION section.

All Version 1.1 Intrinsic UNITs must be in the special library file named SYSTEM.LIBRARY on the "boot" volume at the time a program using the Intrinsic UNIT is executed. Otherwise, the Intrinsic UNIT cannot be used. It is possible on the Apple II to compile a program with the $U option while the Intrinsic UNIT is in any Library File - but that program cannot run until the Intrinsic UNIT is in *SYSTEM.LIBRARY. (The * stands for "boot" drive.) *SYSTEM.LIBRARY can be thought of as a "super" Intrinsic UNIT. Its presence on the boot drive is expected by the system and its capabilities exceed any other Library File. Version IV Pascal does not have this limitation on location of Intrinsic UNITs.

Each UNIT initially compiles into a Library File containing one UNIT (two if a DATA segment is generated). UNITs may be combined (filling more of the 16 available slots or locations for potential program segments) only by using the program named LIBRARY. This program allows creation of a new *SYSTEM.LIBRARY file or any other Library File from previously-compiled UNITs. Learning how to use the program LIBRARY is part of understanding the UCSD Pascal Language System library facility.

Segmentation

The entire UCSD Pascal Language Operating System is divided into parts called Segments. The Regular UNIT and Intrinsic UNIT are both Segment types which together make possible the UCSD Library Files.

Segmentation also allows combining separately compiled code files into one program. This is allowed even when program parts originate in different languages. FORTRAN can use Pascal and Assembly Language parts; Pascal can use FORTRAN and Assembly Language parts.

When a code segment is produced by SYSTEM.ASSMBLER (used for machine language routines), it is called an External Segment. An External Segment must be manually linked. Its code is duplicated and merged into the using segment so that the original code file is not needed on-line after linking. FORTRAN's $EXT declaration is used to include External Segments. Pascal does the same by following the Procedure or Function declaration with "EXTERNAL;".

A segment which can be swapped in and out of memory is called an Overlay Segment. Pascal defines an Overlay Segment by beginning the Procedure or Function heading with "SEGMENT". In FORTRAN, the word "OVERLAY" must end the $USES declaration. Each SUBROUTINE and FUNCTION in an Overlay Segment can be called by FORTRAN, while Pascal sees only the outermost PROCEDURE or FUNCTION. FORTRAN can declare any acceptable Regular UNIT (defined below under "Organizing Intrinsic UNITs") as an Overlay Segment. Apple II Pascal can declare all Intrinsic UNITs as Overlay Segments with a no-load compile-time option. The code for an Overlay Segment is loaded into memory only when it is called. That area of memory is marked as unused when the Overlay Segment is no longer active.

LIBINFO

The above Segment types are summarized as follows:

| SEGMENT | LOCATION | CHARACTERISTICS |
|---|---|---|
| Regular | Any Library File | Must be linked; code is duplicated |
| Intrinsic | *SYSTEM.LIBRARY during use | No linking or code duplication |
| External | Assembly Language Code File | Must be linked; code duplicated & merged |
| Overlay | Pascal: Included with program code file<br>FORTRAN: Any acceptable Regular UNIT | Pascal: routine beginning with SEGMENT<br>FORTRAN: $USES followed by OVERLAY |

There are thirty-two Segment numbers available during execution of a Version 1.1 Pascal Operating System program. The Operating System refers to Segments by disk location (not by name) after execution begins. Disk locations are assigned numbers 0 through 31, of which numbers 7 through 15 may be used to segment the program itself, exclusive of Intrinsic UNITs. Details of these segmentation rules are as follows:

1. The system considers itself as Segment #0.

2. The program being executed is always Segment #1.

3. Segment numbers 2, 3, 4, 5, and 6 are for system use.

4. The first possible segment number for assignment is Segment #7. If a program USES a Regular UNIT, it will be Segment #7. Assume the program uses a total of three Regular UNITs and is itself broken into two Overlay Segments and one External Segment:

   a. First, the Regular UNITs get numbers in the same order that they are listed in the USES statement. In this case Segment #7, Segment #8 and Segment #9 are assigned to the Regular UNITs.

   b. Next, the Overlay Segments are assigned numbers. For this example, they become Segment #10 and Segment #11.

   c. Unique to all segment types, External Segments do not require segment numbers. When linked, an External Segment is merged with the segment it is linked with and no longer has its own identity (it is no longer a "segment").

5. If this program USES five Intrinsic UNITs, are they assigned Segment #12 through Segment #16? No, because an Intrinsic UNIT must be assigned a number when it is created. Intrinsic UNITs given numbers below the range 16 through 31 may conflict with program segment numbers. In this example, an Intrinsic UNIT numbered lower than 12 could not be used.

6. Apple II Pascal users may break rule 4 by using the "next segment" compile-time option explained on page 11 of the

LIBINFO

"Addendum to the Apple Pascal Language Reference Manual". Assigning a segment's number may become necessary in order to use FORTRAN subroutines in Pascal programs (see RTUNIT below), or to overcome Intrinsic UNIT number conflicts such as the one noted in Rule 5.

Note that the above applies ONLY to Apple Pascal Version 1.1 and that newer versions provide for more segments which, along with improved memory management, allow for creation of larger programs.

## Organizing Intrinsic UNITs

Intrinsic UNITs can have segment numbers that conflict with each other. Purchased library routines can come with conflicting Intrinsic UNIT numbers which cannot be changed. This can force a decision to reassign numbers on Intrinsic UNITs for which the user has source listings. If an Intrinsic UNIT is renumbered, all programs using that UNIT must be recompiled.

Intrinsic UNIT segment numbers must be different to be used in the same program. Likewise, the Intrinsic UNIT segment number must be different from any numbers assigned by the system to the program. Since no Apple Pascal Version 1.1 segment numbers higher than 15 can be used by non-Intrinsic program UNITs, it is best to give all *SYSTEM.LIBRARY Intrinsic UNITS high numbers, preferrably in the range of #16 through #31.

There is no problem if *SYSTEM.LIBRARY has two or more Intrinsic UNITs with, for example, Segment #17. This is because segment numbers exist only for the program that happens to be running at the moment. Only one #17 may be used by the program. The system can tell which #17 of several is being referred to because the Library's slot number is used by the system (rather than the Segment number).

Apple II Pascal uses Intrinsic UNITs to perform some operations, effectively reserving additional segment numbers for the Operating System. This is not true if a program using *SYSTEM.LIBRARY does not use the feature. FORTRAN does not require any of these Intrinsic UNITs:

| SEG # | INTRINSIC UNIT | OPERATIONS SUPPORTED |
|---|---|---|
| 29 | TRANSCEND | Transcendental functions (SIN,COS,EXP,ATAN,LN,LOG,SQRT) |
| 30 | LONGITI | LONG INTEGER data type |
| 31 | PASCALIO | REAL data type |

All Global Procedures, Functions, Types or Variables, whether defined in the INTERFACE part of a UNIT or globally in the program itself, must have unique names. A UNIT is not "acceptable" to FORTRAN if it declares a Global Type or Variable or if all public data types are not recognizable. Also, FORTRAN uses different I/O conventions and multidimensional array ordering.

LIBINFO

## FORTRAN Segment Assignments

The segment assignments for FORTRAN at first seem to violate segmentation rules but in fact do not. A FORTRAN program is actually compiled as a Regular UNIT of a Pascal program named MAINSEGX. A Pascal program is always Segment #1. The FORTRAN program is downgraded to a Regular UNIT and becomes Segment #7 (Rule 4 above).

Every FORTRAN program is required to use the Regular UNIT named RTUNIT, a very hefty set of routines sometimes located in *SYSTEM.LIBRARY. Thus, RTUNIT becomes Segment #8. If a Pascal program uses a Regular UNIT that is FORTRAN, the programmer must insure that RTUNIT is used and assigned as Segment #8 during compilation or errors may occur during execution of the FORTRAN routine(s).

## Segment Assignments with *SYSTEM.LIBRARY

The author's organization of his normal *SYSTEM.LIBRARY gives no Segment number or other conflict among Intrinsic UNITs, so that all may be used at once if needed. The following table shows the author's slot assignments:

| SLOT# | SEG # | NAME |
|---|---|---|
| 0 | -- | RTUNIT |
| 1 | -- | expansion |
| 2 | 18 | BPBSTUFF data |
| 3 | 19 | CLOCKSTUFF |
| 4 | 20 | UOUT |
| 5 | 21 | NUMBERIN |
| 6 | 22 | APPLESTUFF |
| 7 | 23 | APPLEGRAPHICS |
| 8 | 24 | APPLEGRAPHICS data |
| 9 | 25 | MODMSTUFF |
| 10 | 26 | STRINGSTUFF |
| 11 | 27 | BPBSTUFF |
| 12 | 28 | CHAINSTUFF |
| 13 | 29 | TRANSCEND |
| 14 | 30 | LONGINTI |
| 15 | 31 | PASCALIO |

RTUNIT (if present) is the only Regular UNIT in *SYSTEM.LIBRARY, leaving only one slot available for expansion (because 15 slots are used). RTUNIT is put in *SYSTEM.LIBRARY purely for the convenience of "automatic" linking. This is a waste of valuable boot disk room unless a Corvus or other large-capacity boot drive is in use. RTUNIT can be stored on the same diskette with the FORTRAN compiler as a Library File and linked manually. FORTRAN programs are often built up from Regular UNITS so giving up automated linking is seldom noticed.

In the following table of possible program segment assignments, capital letters indicate a UNIT name in *SYSTEM.LIBRARY which should be in the USES statement of Pascal (first 8 letters are significant)

LIBINFO

or the $USES statement of FORTRAN (if that Intrinsic UNIT is acceptable to FORTRAN). Some UNITs do not require a USES reference. RTUNIT requires a USES statement only from a Pascal main program, while LONGINTI and PASCALIO cannot be in a USES statement because they are for system use.

"Undefined" means that a segment number can be used by a Regular UNIT or Overlay Segment as encountered by the compiler. "Not used" means that only an Intrinsic UNIT can use this segment number but no such Intrinsic UNIT is in the Library. Items identified by an asterisk ( * ) can be redefined because the author has the source listing(s), though UOUT cannot be assigned a different Intrinsic UNIT number because it is used by an Intrinsic UNIT with no source listing (APPLEGRAPHICS).

| Seg # | Name | Seg # | Name |
|---|---|---|---|
| 0 | system | 16 | not used |
| 1 | program | 17 | not used |
| 2 | system built-ins | 18 | * BPBSTUFF data |
| 3 | system built-ins | 19 | * CLOCKSTUFF |
| 4 | system built-ins | 20 | * UOUT |
| 5 | system built-ins | 21 | * NUMBERIN |
| 6 | system built-ins | 22 | APPLESTUFF |
| 7 | undef or FTRN prgm | 23 | APPLEGRAPHICS |
| 8 | RTUNIT if FORTRAN | 24 | APPLEGRAPHICS data |
| 9 | undefined | 25 | MODMSTUFF |
| 10 | undefined | 26 | STRINGSTUFF |
| 11 | undefined | 27 | * BPBSTUFF |
| 12 | undefined | 28 | CHAINSTUFF |
| 13 | undefined | 29 | TRANSCEND |
| 14 | undefined | 30 | LONGINTI |
| 15 | undefined | 31 | PASCALIO |

When examining a similar library with either of the programs LIBRARY or LIBMAP, notice that RTUNIT (if present) is listed with a Segment #7 rather than #8 as it must be to function in a program. The #7 has no meaning because this Regular UNIT has a segment number assigned as the program is compiled.

The above example of *SYSTEM.LIBRARY has one major change (other than additions) from the standard *SYSTEM.LIBRARY discussed in Apple II documentation: TURTLEGRAPHICS and its data segment have been replaced by an enhanced (three-dimensional) graphics package named APPLEGRAPHICS. The replacement does not support polar coordinates but otherwise allows easy conversion of existing programs using graphics. In addition to offering three dimensions, APPLEGRAPHICS will drive an X-Y plotter through UOUT, an Intrinsic UNIT which requires its IMPLEMENTATION to be programmed by the user for the system plotter. Intrinsic UNIT IMPLEMENTATIONs but not INTERFACEs may be revised without recompiling using programs.

LIBINFO

Overlaying

With a lot of disk storage, brute-force chaining of programs using common data files might handle a large-memory application on a small computer. This is the only method provided by most small computer Operating Systems. However, there is a much better approach to solving the memory crunch. It is called overlay programming.

An understanding of Library Files and especially of segmentation is vital to successful organization of a very large program to run on a small computer. Physically, overlaying means that portions of the computer's working storage (RAM) are being periodically overwritten by segments of code copied in from disk. Therefore, the programmer is balancing available working storage against execution speed, all within the limits of the rules for program segmentation.

Data overlaying also occurs. Room for data can limit an application despite program overlaying, sometimes requiring careful use of properly managed disk data files and other techniques. As a first consideration, the programmer should use the minimum possible number of global variables so that each Overlay Segment includes its own data in the form of local variables. Good data base subroutines (such as PFAS by C.J. Wigglesworth Software [now Gryphon Systems], a Regular UNIT) help manage the data problem.

If no Regular UNITs are used (FORTRAN uses two), up to 9 Overlay Segments can be defined in the program (#7 - #15). Apple Pascal can also overlay Intrinsic UNITs for Segments #16 through #31 by using a no-load compile-time option, but Segments #29, #30 and #31 may be reserved for system use as TRANSCEND, LONGINTI and PASCALIO.

After a program has been adequately segmented to execute within the available working memory, the programmer should review program and data organization to minimize disk activity - the frequency of overlaying. Apple Pascal offers the Resident compile-time option ($R) for keeping Overlay Segments in memory between calls. This is a valuable option when an Overlay Segment call is in a loop.

One fact that may be overlooked is that overlaying is not nested. If a Pascal Segment Procedure contains or calls a Segment Function, for example, the Segment Procedure is loaded into memory and executed up to the point of the Function call. Then the Segment Function is loaded over the same approximate area in memory and executed. Finally, the Segment Procedure is reloaded before it can complete execution. This can be exceedingly inefficient. Note: Version IV Pascal improves memory management by a remarkable degree, so swapping in and out of memory occurs only when necessary.

LIBINFO

Summary

The proper way to use a Library File is to store in it all of the related routines which may be repeated in other programs. For Version 1.1 and earlier, Intrinsic UNITs should be reserved for the largest or most frequently repeated routines. Remember that Intrinsic UNITs must be used from *SYSTEM.LIBRARY for Version 1.1. The author encourages programmers to use External Segments only in Intrinsic UNITs so that conversion to another computer will consist only of rewriting the IMPLEMENTATION part of some *SYSTEM.LIBRARY segments.

If *SYSTEM.LIBRARY is exchanged among several special application libraries, the user must be careful when using any programs which expect UNITs from *SYSTEM.LIBRARY. For example, the author's *SYSTEM.STARTUP must find CLOCKSTUFF and BPBSTUFF there in order to run. Any changes in *SYSTEM.LIBRARY are unknown to the system until the user reboots the system.

Documentation of the various library UNITs may not be readily available. However, all user-accessible information on a Library File can be reviewed at any time by using LIBMAP to read the INTERFACE portion of any Library File.

Learning how to use the Linker and the utility program LIBRARY, how to construct UNITs, how to document a Library File with the utility program LIBMAP, where to put needed files on existing disk drives and related skills should be a little easier armed with this background. The UCSD Operating System and the Pascal Language offer a lot of power and sophistication in an economic package. A shortcoming of Pascal Version 1.1 and earlier is lack of extended precision for real numbers. However, Version IV offers 12 digit precision and is (potentially) more portable than Apple Pascal 1.1.

Where does the future lie? ADA, a language whose development was financed by the U.S. Department of Defense, is a facinating "superset" of UCSD Pascal. ADA overcomes certain shortcomings of Pascal which may not be apparent to the casual programmer. With minimum memory sizes approaching 128k bytes, the author expects ADA to become the world standard language, with Pascal programmers having a definite head start. In fact, Pascal programmers have a definite advantage today. The reader's comments concerning this article or using the UCSD Pascal Language System are welcomed.

Ben P. Baston, President
Southwest Computer Consultants, Inc.
7870 West Bellfort
Houston, Texas 77071
(713) 777-3188

# DEALING WITH DOS

By

Clark Johnson

Part #1

This is the first installment in (what I hope to be) a monthly column on Apple DOS. My intention is to start out with the basics of DOS and work up in complexity as the months progress. In the beginning some of the experienced people might find the column trivial, but please hang in there. I hope to eventually cover items of interest to everyone.

I would like to have feedback - from both sides of the fence. Novices, please let me know what you don't understand or would like to see covered. "Experts", if I make a mistake or have overlooked something of importance, I need that info also.

Call the HAAUG hotline (895-8612) to leave a message for me. Be sure to state that the message is for the column "DEALING WITH DOS". If you need feedback immediately, I will try to call you with the information. Otherwise, I will attempt to cover your question or comment in the next newsletter. If you are unable to get to the hotline, you could call me at home (370-3543).

## At The Beginning

The Apple computer (the Apple I as it's called today) was first sold in 1976. As we all know, the Apple was the first complete microcomputer system available. Prior to its debut, the computer enthusiasts had to construct their own by buying the necessary hardware. However, these early personal computers did not have any convenient data or program storage devices. As a matter of fact, when Steve Wozniak was first breadboarding the prototype Apple, he had to type in the all his Basic language hex codes every time he used the machine, prior to doing anything constructive. This process could take a couple of hours before he was ready to start demonstrating the machine.

Fortunately, one of the earliest improvements to the Apple prototype was the development of a system to handle a tape recorder to store or retrieve data. The tape recorder interface was on the first Apple I's sold. The tape recorder was a vast improvement over previous methods (typing or paper tape punch). But the tape recorder left much to be desired. When the Apple ][ was introduced, it too was limited to tape storage.

The personal computer probably would have never made it off the ground if we had to live with the tape system. Its reliability was less than perfect and speed was slow, but the main problem was in its data retrieval and storage. A disk can be randomly accessed -- that is, any program (file) on it can be directly accessed and pulled into the computer's memory. Also, one file can call on another file for information, data retrieval, or for help in calculations. Contrast that to a tape system that is sequentially accessed. To load one file into the computer's memory, you must first run the tape over all files stored in front of the file you wish to access. Obviously, you can't effectively manipulate data back and forth between files. Also, you could not name a program in tape storage. You had to keep records of the tape counter values where each program was stored.

Apple introduced the Disk ][ in June, 1978 to overcome the shortcomings of the tape system. To control the disk system, Apple released the first version of DOS , entitled DOS 3. (I guess DOS 1 and DOS 2 were pre-release versions.) DOS is an acronym for 'Disk Operating System'. Please note that the word is DOS, pronounced like "boss". It is not pronounced like doze, dose, or d-o-s.

A number of bugs quickly surfaced in the first version, so only a month later in July of 1978 Apple released DOS 3.1 to correct some of the more obvious problems. DOS 3.2 was introduced about six months later in February, 1979. This DOS version was basically the same as the previous one (about 90% the same according to "Beneath Apple DOS"), but several improvements were made over the older version. DOS 3.2 was an essentially bug-free system. but some minor corrections were made in July, 1979 and released as DOS 3.2.1.

Typically, when people refer to DOS 3.2, they are talking about this DOS 3.2.1. Another name commonly used for this version (and really all previous versions) of DOS is "13-sector DOS". DOS 3.2.1 remained the primary system in use for a little over a year.

However, in August, 1980, Apple introduced DOS 3.3 (or 16-sector DOS). Its main advantage over DOS 3.2 was that the new version could store about 23% more data on a disk. We will later explore the mechanism of disk storage and see why DOS 3.3 is more space efficient than DOS 3.2.

If you are a relatively newcomer to the Apple world, it is possible that you have never used DOS 3.2. DOS 3.3 has been around for over two years now, and it is also unlikey that future significant revisions to Apple DOS will occur.

## What Is A DOS ?

As I said earlier, DOS is an abbreviation for 'Disk Operating System'. DOS is nothing more than a computer program that lives inside your Apple along with your other program(s). It is a fairly large machine language program (about 37 sectors) that supervises and controls all functions relating to disk storage and retrieval. It monitors every action that you take (like a Big Brother) to be able to step in and take control whenever a computer command requires use of the physical disk system. In addition to the disk control functions, DOS also adds a couple of elements to Basic and machine language programs that would not be otherwise possible.

I would think that practically everyone reading this article owns a 48K machine (ignoring RAM cards for the present). Two or three years ago there was a wide variety of memory sizes, but today computer memory is so cheap that it's really foolish not to own a full size machine. Therefore, all future discussions will assume a 48K machine unless specifically indicated otherwise.

DOS then occupies about 21% of your computer's memory capacity (DOS is about 10K long). Most of the rest of your machine is available for your computer programs. Future discussions will elaborate on the location of DOS in your computer.

How Does DOS Get Into Your Computer ?

DOS resides in RAM memory (random access memory). RAM memory is "volatile". All computer coding that resides in volatile RAM memory is lost whenever the machine is powered down, because an electric voltage is required to keep the coding "alive". Therefore, whenever you turn your machine on, DOS must be loaded into the computer's memory. This is done by "booting" a disk that contains DOS. All normal disks have DOS located on the first three tracks on the disk (out of a total of 35 tracks on the disk). The "boot" process is started whenever the computer is turned on (assuming an Apple ][+) with a normal disk in the No.1 disk drive, or whenever PR#6 <ret> or IN#6 <ret> is typed in after a Basic language prompt (] or >). If your computer shows the monitor prompt (*), typing in 6 ctrl-P <ret> or C600G will boot DOS from a disk into your computer. It takes about 3-4 seconds to load DOS into your computer.

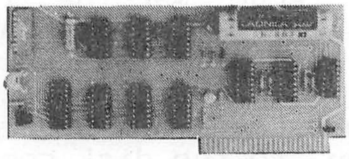
The word "boot" comes from the act of bootstrapping your disk operating system to an on-line condition. The boot process uses only a small amount of "hard-wired" coding to bring DOS into the computer. This coding is located in two chips on the disk controller card. During the boot process DOS loads parts of itself into the computer and then uses the previous part to load a larger portion of DOS, thereby "bootstrapping" itself on-line. The boot process is complicated enough to warrant a significant amount of discussion and will therefore be deferred to a later date.

Let's try some hands-on experience to prove some of the prior comments. First, turn on your computer without a disk in Drive 1.

# APPLE PERIPHERALS ARE OUR ONLY BUSINESS

## TIME II

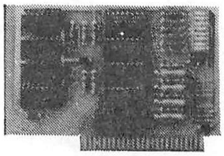
### THE MOST POWERFUL, EASIEST TO USE CLOCK FOR YOUR APPLE



- ● Time in hours, minutes and seconds.
- ● Date with year, month, day of week and leap year.
- ● Will enhance programs for accounting, time and energy management,
- ● remote control of appliances, laboratory analysis, process control, and more.
- ● 24-hour military format or 12-hour with AM/PM indication.
- ● User selectable interrupts permit foreground/background operation of two programs simultaneously.
- ● Crystal controlled for .0005% accuracy.
- ● Easy programming in basic.
- ● On board battery backup power for over four months power off operation (battery charges when Apple is on).
- ● Twenty-seven page operating manual included with many examples of programs to use with your Apple in any configuration.
- ● Includes disk containing a DOS Dater and many other time oriented utilities plus over 25 user contributed programs at no extra cost.

**PRICE $129.00**

## SUPER MUSIC SYNTHESIZER



- ● Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo and boot the disk supplied and you are ready to input and play songs.
- ● It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- ● We give you lots of software. In addition to Compose and Play programs, the disk is filled with songs ready to run.
- ● Easy to program in basic to generate complex sound effects.
- ● Four white noise generators which are great for sound effects.
- ● Plays music in true stereo as well as true discrete quadrophonic.
- ● Envelope control.
- ● Will play songs written for ALF synthesizer (ALF software will not take advantage of all the features of this board. Their software sounds the same in our synthesizer.)
- ● Automatic shutoff on power-up or if reset is pushed.
- ● Many many more features.

**PRICE $159.00**

## ANALOG TO DIGITAL CONVERTER

- ● 8 Channels
- ● 8 Bit Resolution
- ● On Board Memory
- ● Ratiometric Capability
- ● Fast Conversion (.078 ms per channel)
- ● Eliminates The Need To Wait For A/D Conversion (just PEEK at data)
- ● A/D Process Totally Transparent to Apple (looks like memory)

The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on board memory at the end of each conversion. No A/D converter could be easier to use.

Our A/D board comes standard with 0, 10V full scale inputs. These inputs can be changed by the user to 0, -10V, or -5V, +5V or other ranges as needed.

The user connector has +12 and -12 volts on it so you can power your sensors. (These power sources can be turned off with on board dip switch).

| | | | |
|---|---|---|---|
| Accuracy | 0.3% | Input Resistance | 20K Ohms Typ |

A few applications may include the monitoring of ● flow ● temperature ● humidity ● wind speed ● wind direction ● light intensity ● pressure ● RPM ● soil moisture and many more.

**PRICE $129.00**

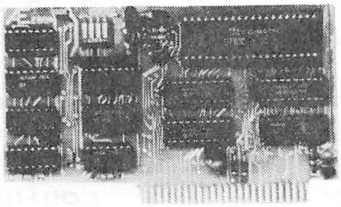
## DIGITAL INPUT/OUTPUT BOARD

- ● Provides 8 buffered outputs to a standard 16 pin socket for standard dip ribbon cable connection.
- ● Power-up reset assures that all outputs are off when your Apple is first turned on.
- ● Features 8 inputs that can be driven from TTL logic or any 5 volt source.
- ● Your inputs can be anything from high speed logic to simple switches.
- ● Very simple to program, just PEEK at the data.
- ● 4 other outputs are also provided. User 1, reset, interrupt request, non-maskable interrupt.
- ● Now on one card, you can have 8 digital outputs and 8 digital inputs each with its own connector. The super input/output board is your best choice for any control application.

**PRICE $62.00**

## Z-80 CARD

- ● TOTALLY compatible with all CP/M software.
- ● Executes the full Z-80 and 8080 instruction set.
- ● Allows you to run your Apple CP/M based programs.
- ● Does EVERYTHING the other Z-80 boards do, plus supports Z80 Interrupts.
- ● Hardware and software settable switch options.
- ● An on-card PROM eliminates many I.C.'s for a cooler, less power consuming board.
- ● Complete documentation included. (user must furnish software)



**PRICE $139.00**

Since our inception, Applied Engineering has continually expanded its line of Apple peripherals bringing you easy-to-use designs.

We are the innovators not the imitators. Utilizing state-of-the-art technologies. Applied Engineering is continually improving its products. The above represents our most recent developments. Applies Engineering offers you the highest quality peripherals at the lowest possible price.

Applied Engineering's products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a one year warranty.

**All Orders Shipped Same Day.**
**Texas Residents Add 5% Sales Tax.**
**Add $10.00 If Outside U.S.A.**

Send Check or Money Order to:
**APPLIED ENGINEERING**
**P.O. Box 470301**
**Dallas, TX 75247**

**See Your Dealer**
**or Call (214) 492-2027**
**7 Days a Week**
**Master Card & Visa Welcome**

The drive will come on and stay running, trying to find a disk it can read. Stop the drive by pressing RESET. You will get the Applesoft prompt (]). You now have a fully operable computer (but without a disk operating system). You could type in a program and run it. (Try a small program to prove this.) But you can't save to a disk or load from one. Now place a disk into Drive 1 and type "CATALOG". You will get a "?SYNTAX ERROR" message. This is because the CATALOG command is a DOS command, and not part of the computer's built-in command list. The CATALOG command (and many others) are loaded into the computer when DOS is booted. In this example, you did not have DOS loaded when you typed "CATALOG".

To continue our experiments, let's try to boot DOS by using several different methods. There are two obvious methods - by either turning on the computer (Apple ][+) or by typing PR#6 with the computer already on. Another less obvious way is to type IN#6. Try that. Now try typing CALL -14848 <ret>. Or CALL 50688 <ret>. Now let's try booting from the monitor. Type CALL -151 to get into the monitor. The "*" monitor prompt will appear. From the monitor, the following commands will cause the computer to reboot: C600G <ret> and 6 ctrl-P <ret>. That's probably enough to get the point across. Don't worry if you don't understand why all those different commands work. That's one of the things we will study as time progresses.

One more word about the term "boot" -- that is, you boot a disk or you boot DOS. You do not boot a program. I've had some of my friends new to the computer world use the term "boot a program" when they mean load or run a program. This may sound picky, but it's all part of learning about DOS.

### DOS Procedures

There are 20 valid DOS commands, but this month we will examine only 4 of them. These are :

1. INIT
2. RUN
3. LOAD
4. SAVE

Let's take INIT first. INIT is a command that causes a new disk (or an old disk that has worthless data) to be initialized or formatted with certain data marks that DOS uses to recognize where it is storing data on a disk or where to retrieve data from a disk. A new disk has absolutely no identifying marks on its surface. If you put a new disk into your drive after DOS is booted and type "CATALOG", you will get nothing but a grinding noise and a "SYNTAX ERROR", because DOS can't recognize any data on the disk. Also, if you try the same thing with a copy-protected disk, you will get the same result. Copy-protected disks are obviously initialized, but with non-standard data marks that your DOS can't recognize.

Initializing a disk is very simple. First you need to boot up your computer with a "virgin" copy of DOS. By this I mean use a disk that has a DOS that has not been altered by something. There are many ways you can alter DOS, but you don't want strange DOS's on your disk unless you know exactly what you're doing. The System Master that came with your computer would be a good choice, but is not necessary if you have other disks with clean DOS's.

Now you have of choice of which "hello" program to put on your disk when it is initialized. The hello program is the one that will automatically run when your disk is booted. The program does not actually have to be named "HELLO", but it is usually good practice to do so, since you will never be confused in the future about what is actually your hello program on a particular disk. Now load or type in your hello program (after booting on the clean DOS). You may load into the computer's memory a program from another disk or type your own; or if you really don't want a program to run when you boot your disk, just type "NEW" to clear out of memory any program there. In this case the new disk, when booted, will still try to run its hello program, but since the blank hello program has no instructions, nothing will happen except that you will get the ] prompt.

Below is a very simple hello program that you might wish to use. It does not do anything fancy - it gives the date when the disk was created and then lists a CATALOG of the disk's contents.

```
10 HOME:HTAB6:PRINT "DISK INITIALIZED ON 10/10/82"
20 PRINT CHR$(4)"CATALOG"
```

The club library has a almost full disk that has nothing but hello programs (#1A2). These hello programs have fancy ways of bringing the computer on line and giving the CATALOG for program selection. Get the disk at the next meeting and try out the different ways of bringing a disk up.

The simplest way to initialize the new disk would be to then insert the new disk into Drive 1 and type INIT HELLO <ret>. The red light on the drive will come on, and the initializing will take place for about one minute. The first phase of initializion consists of putting the identifying marks on the proper areas of the disk. Subsequent phases a) build a catalog track on the disk where the file names will be stored, b) store DOS onto tracks 0, 1, and 2 and c) save your hello program that was in memory. All of this takes place without any more assistance from you. You now have a disk ready to store more files.

At this point it might best be to digress to discuss the type of DOS that has been stored on your new disk. It won't be exactly the same as the DOS from your Master Disk, even if you had used the Master to boot up initially. You have instead created a "slave" disk as opposed to a "master" disk. This used to be a fairly important distinction when there were many different sizes of Apple memory. A master disk, when booted, will perform a check to see what size (how much RAM memory) computer is being used. It will then place the DOS program in the "highest" part of the computer that is available, thereby leaving the maximum amount of memory below DOS for your programs. A slave disk will not check for memory size and will always place DOS into memory where it was located when the disk was initialized. For example, if a disk had been initialized on a 32K machine and then used to boot a 48K machine, then the 48K machine will act like a 32K computer because DOS will be located lower than desirable in the larger machine. 16K of memory will be sitting idle above DOS and would be therefore wasted. Going the other way, if the disk had been initialized on a 48K machine and then used to boot a 32K computer, you would really have a problem. The boot process would try to place DOS at a spot that doesn't exist on the smaller memory size. The system would hang, and the boot would fail.

Now that sounds like a potentially severe problem, but not really. Apple computers with less than 48K memory are about as rare as abacuses today. The odds are almost 100% that any disk you use was initialized on a 48K machine and that any disk that you initialize will always be used on 48K computers. If you really want to convert your slave disks to master disks, there is a program (Master Create) on the System Master disk that will do just that. I personally feel that it is unnecessary and really could be a disadvantage because a master disk "walks on" more memory when booting. Again, this is an issue for a later chapter when we discuss memory utilization in the computer.

Final comments are necessary regarding the INIT command. When you initialize a disk in the simple manner that I explained earlier, you will always have a disk with a volume number of 254. If you desire to have a disk with a different volume number or would like to initialize from a different drive or slot, then use the following command as an example:

```
INIT HELLO,S5,D2,V36
```

In this example, the disk initialized will be in Drive 2, which is controlled by a disk controller card located in Slot 5 (instead of the more normal Slot 6). It will have a volume number of 36. You may use all or part of the parameters given in the above example (i.e., INIT HELLO, V36 is OK). Again, it is not necessary to actually name your hello program "HELLO", only preferable in most cases.

Also, you might be wondering about compatability with systems that have 64K or more memory by using RAM cards. Well, usually any sytem that relocates DOS into the RAM card has the INIT function disabled. In other words, don't worry about the fact that you don't have a RAM card and other people might (or vice-versa). You can still keep all your disks as slave disks.

RUN is the second DOS command to explore. You may wonder why RUN is a DOS command, because it is also an Applesoft command (part of the built-in commands of the Apple). That's why you could write and run a program when DOS was not in the computer, as in our earlier example. However, RUN must also be a DOS command because it is used to first load and then execute a program from a disk, which is DOS's domain. DOS actually intercepts the RUN command when you type it in to determine whether it needs to load a program in from a disk or whether to just execute the program already in memory. RUN can be used with the same drive, slot, and volume parameters as in INIT.

LOAD and SAVE are the last two DOS commands to be discussed this month. And they are two of the simplest. LOAD is used to transfer a Basic (either Applesoft or Integer) program from a disk to the computer's memory. Example :

```
LOAD HELLO, D2
```

will load into the computer's memory a program called HELLO which is on a disk in Drive 2. This command will not execute the program; it will not run unless the command RUN is specifically typed in. LOAD is beneficial for several reasons. You might want to examine the listing of the program without actually running it, or you might want to modify the program before running it. Also, LOAD is useful when used in conjuction with SAVE.

SAVE is the opposite of LOAD. It will save onto a disk the Basic program which is currently in the computer's memory. Obviously, the program must be given a name, as in the following example:

```
SAVE TEST,D3,S5
```

This will save a program currently in the computer's memory and give it the name "TEST". It will be saved onto a disk which is in Drive 3, connected to a controller card located in Slot 5. LOAD and SAVE, when used together, provide one means of transferring a program from one disk to another. You first LOAD the program into memory from one disk and then SAVE it to another disk. You may use the same name if you wish, but it is not necessary.

A final comment on the DOS commands given in this article (and true for future DOS commands) -- you do not have to specify the Drive, Slot, or Volume. However, if you do not, the Apple will automatically assume that the new command will have the same values for drive, slot, and volume as the last command given. Or the values may have been implied by a boot. In this case all values will be the same as on the boot (Except you don't have to specify volume number).

Next month we will finish up on the DOS commands and then get into DOS utiliies, specifically the ones that come on the System Master disk.

To repeat, I would appreciate feedback and hints on preparation of this column.

## P.S. - PLEASE NOTE

I just today got word on something I've been trying to track down for a couple of weeks. I have been wanting to get access to a good disk read and write program to circulate for those readers of this column that really want to learn about DOS and disk formatting. A good disk analyzer program is an absolute must for any of you that are interested. There are several on the market, but one of the easiest to use and one of the most flexible is a program called DISKFIXER.

Unfortunately, DISKFIXER was no longer being sold due to competition from programs such as INSPECTOR, WATSON, and BAG OF TRICKS. While these are all very good and offer several features that DISKFIXER does not have (I know because I have all of these programs), DISKFIXER is nevertheless the best all-around disk analyzer program for those people new in this field. I was able to track down the company that held the liscence to DISKFIXER. The representative stated that he had an ample supply of these programs (with full documentation) available. The original list price was $30, but he will let us have all we need for $10 each. This is an outstanding buy, and I encourage each of you to buy this program. Please let me know (370-3543) if you want a copy. This is an offer you shouldn't refuse.

-----==*==---

## CALL FOR ARTICLES

Articles and program listings should be submitted in hardcopy form, and, if possible, on disk in Applewriter ][ or ///, Apple DOS or SOS text, Professional Easywriter, Wordstar, Palantir ][ OR ///, or Pascal files, or via modem (358-6687). If not submitted on disk, articles must be free of typing or spelling errors as they cannot be retyped. Diskettes will be returned to the author provided his name and address are on them. Printed material should be printed using normal size characters, a new ribbon and enhanced print if your printer is so equipped. Margins should be set at 7 and 73 for articles. Listings should be printed in 40 column mode and should be printed in compressed print. Thermal paper should be avoided because it does not reproduce well. Authors of published articles will receive a blank diskette per page as compensation. If the last page of an article is a half page or longer it will qualify for a diskette. The Apple Barrel reserves the sole right to determine which articles are used. Submit articles to:

Houston Area Apple Users Group
Apple Barrel
2218 Running Springs
Kingwood, TX 77339

## WANT/DON'T WANT ADS

FOR SALE: Data Products serial thermal printer, 80 cps, quiet. Includes cable. Asking $300.

Call Lee Gilbreath at 342-2685.

---===*=====---

WANTED: ROMwriter or equivalent.

FOR SALE: Airline-proof case for Apple ][ and two drives. Never used. Lists for $175+tax, sell for $100. Non-airline-proof case. Lists for $125, sell for $50.

Call Mike Kramer at 358-6687

---===*=====---

FOR SALE: VISICALC 3.3 with original documentation, $75.

Call Cy Helm at 676-5276 (work) or 974-5153 (home)

---===*=====---

FOR SALE: Mountain Computer CPS Card with all accessories, cables, software including DOS, CP/M, and Pascal. New condition. Lists for $339. Will sell for $125.

Call Michael Flinn at 667-1869

---===*=====---

# GAME REVIEWS
By Bill Muhlhausen

## SOUTHERN COMMAND
Product of Strategic Simulations

This game is a recreation of the Yom Kippur War crossing of the Suez Canal. The instruction book is very detailed and will require 30 to 60 minutes study before attempting to play the game.

SSI has presented the people who enjoy detailed board type war games a very entertaining program. The game may involve either 10 or 20 turns depending on the preference of the player or players. It is a one or two player game. In solitaire play you have four options as to level of difficulty. A solitaire game is faster to play than the two player game. In playing the game in both modes I and my associates found that we could not expect to finish a game in less than 5 to 8 hours. In fact 10 to 12 hour games were not unusual. The program does provide the players the ability to save a game and come back to it. Our experience with this feature was that while the game could be saved it could not be restarted. Four attempts were made to restart a saved game. You are asked to wait. After four waits of five minutes and more we gave up on the idea of attempting to restart the game.

This game is very realistic as to the variables of combat. Terrain effect on movement is considered. Setting of ambushes and hidden movement is possible. Long range artillery has a variable effect on the target fired on. Air strikes are possible by the Israeli forces on any day turn. The Egyptians may twice during the game neutralize an air strike on the next Israeli turn by committing their air force. Terrain also affects the combat effectiveness of all units. Units are able to reorganize in order to regain fighting strength and efficiency. Artillery units may not reorganize. Units may be set up in dugin predefense positions or may be put on defense at any time deemed necessary. You are able to change combat modes from combat to transport to another mode as long as you have enough movement points to do so. You may preprogram a future move of a unit giving players a delayed move ability.

The program does allow players to set up a non-historical game. In playing this game we found that the Egyptians were truly difficult to defeat in both solitaire and two player modes. This is good for the person who prefers to play solitaire because it always provides a challenge that keeps the game from becoming stale. The extremes of always winning or losing seem to have been avoided. The mastering of the various commands took my group 10 to 15 hours of actual playing time. Those of you who are ardent board gamers may be able to shorten this time to some degree. Once the commands are learned, however, the game is fairly easy to play. The graphics are very good. Each side's pieces that are not hidden are easily recognized and limited stacking is allowed. You can see a map of the overall positions but not the individual units so spotted. The movement of pieces may not be watched on the overall map. This movement is shown on localized maps. The ability to center on the unit being moved or to scroll on the horizontal or vertical plane is present. All the graphics are in color.

In conclusion I find that the game is well worth owning provided you are willing to spend the time needed to learn it. Those who want to take a disk out of the wrapping and be proficient in five or ten minutes will not enjoy it.

SERPENTINE
Product of Broderbund Software

Serpentine is a game similar to Pac Man. You are matched against the computer with the object being to eat as many orange snakes from the rear as you can. If you eat all three snakes set against you you go up a level and a different maze is shown on the screen. It is possible to get more than three snakes to attack with by laying eggs. Eggs that are not destroyed hatch to give another attack potential for the player.

This game may be played with either keyboard control or joystick. The keyboard control seems to be much more responsive to commands but more difficult to use than the joystick. The joystick seems a little sluggish to use for fast screen response. The program can be blown when using keyboard control by accidently striking the control key and then a guidance key. The escape key will restart the program where it broke off. You do have the option of defining your own keyboard movement control keys.

Other program options include keeping track of high scores, pause and resume play, switch the joystick axis, and change from keyboard to joystick and back during play.

I found the game to be challenging enough to keep the interest of adults and a definite challenge for younger players. after four hours this reviewer managed level three and 7600 points. The documentation mentions scores of 20000 to 50000 being attainable. My personal opinion is that I like ABM or Pac Man much better due to their faster pace. For those who have patience and enjoy a moderate gaming pace this is truly you cup of tea.

STAR BLAZER
Product of Broderbund Software

Star Blazer is a good quick paced shoot them up type of game. You are assigned five missions. Each of these missions is more difficult than the preceding one.

Some of the features of this game are a finite fuel and bomb supply and the ability to refuel and rearm in flight.

You are given three ships to accomplish all assigned missions. These ships cannot be replaced once destroyed>

Players may use either the keyboard or a joystick to play Star Blazer. The program also allows for a pause and the turning off of the sound effects.

I found this program to be fun to play and very challenging from level three on.

## TUESDAY MORNING QUARTERBACK
Product of EPYX

This sports game is playable against the computer or another player. The instructions are well written and easy to follow. It should not take over five minutes to master all the commands necessary to play the game.

Although the game is easy to learn it has several nice touches. These include setting the length of quarters, a time clock, a 30 second limit on play calling, team statistics at the half and game end, an interesting half time show, random penalties and the ability to trade players between teams. There is also a kicking routine in which the player has to judge when to kick the ball. This is a blessing and a curse as it allows some player control but takes up about 30% of the playing time just to put up the graphic display.

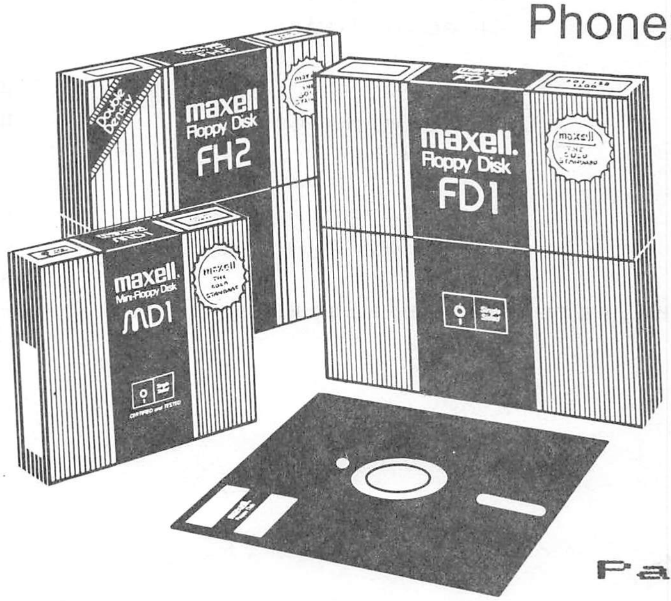
Gamers who expect to see the players move on the screen or be able to control a player movement will not be content with this program. Those who can dispense with these features preferring instead a tough challenge against either the computer or another player should be well pleased upon adding this program to their library. These players will have to call all plays offensive and defensive and worry about injured players.

I wish to express my thanks to Mr. Darrell Smith of the Software Center of Houston, 2200 Southwest Freeway for making available the game software reviewed in this article.

## TROUBLESHOOTING TIPS
## NON-RESPONDING KEYBOARD
By Mike Kramer

Many times I have had someone come to me or call me about a broken computer that seems to boot on power up but will not respond to keyboard input. This is the symptom of a disconnected keyboard cable and can usually be confirmed by an extinguished power light on the keyboard. I say usually because I ran across a system which had the power light on and no keyboard function. After exhausting everything I could think of I disassembled the computer and found the connector on the keyboard encoder board cocked so that contact was made on only some of the pins.

Reconnecting the keyboard cable is not an easy chore, but can be done by most people. First turn off the power but leave the power cord plugged in. Then, after discharging any static from your body by touching the power supply (aluminum box on left) remove all cards from the connectors at the rear of the computer. Turn the computer over, remove the black screws around the edge of the bottom plate of the computer, and lift off the case. Next, carefully squeeze the two plastic standoffs holding on the piggy back encoder board (newer Apples) or remove the 4 screws holding the keyboard on (older Apples). Assure that the ribbon cable connector is fully seated in its connector and replace the the encoder or keyboard as appropriate. Then carefully (you may need a helper) position the case so that you can plug in the other end of the ribbon cable on the Apple's main (mother) board. Screw the case back onto the bottom plate, reinstall the cards in their proper connectors, and see if it works.



# Logo Locus

by Brian Whaley

SECOND IN A SERIES

"Why LOGO?" I am asked. Retorically I respond "Why BASIC?" Usually this leads to interesting academic discussions but not always.

So when I came across Seymour Papert's excellent treatise on computer enhanced learning, Mindstorms: Children, Computers, and Powerful Ideas, I found the answer I was seeking to my "Why BASIC?". At the last Saturday meeting, I made the simplistic statement that I thought Apple Computer Corp.'s book, Applesoft Tutorial, was a good self training guide. That was countered with "It read as though an engineer wrote it." I got defensive stating that I was an engineer. The response to that statement was "Yes, it figures!" Well, they were correct. After all I had taught myself BASIC in college and my sole use of the Applesoft Tutorial was to instruct myself in applying an already known language on a specific machine. Useful, but not a true test of how easily BASIC is learned.

Back to the original question, "Why BASIC?". My college experience should shed some light. First it was available to me and it was the language promoted as the Beginers All Purpose Symbolic Code. I was an engineering student with a technical background. It had a simple instruction set and was interactive. That appealed to my sense of propriety as an engineer.

Papert gives some more clues. In a time when components were expensive, a language that employed as few components as possible (BASIC) was appropriate. Further, it was easily learned by engineers, and appealed to their engineering sensibilities. Engineers design and build computers but today the computer-user world is not populated by engineers, nor are components expensive anymore. The rational behind BASIC must be reexamined especially when children are involved.

Consider an environment where only fifty spoken words are allowed and no further constructions can be made. The variety of human thought would be difficult to express indeed. That is one of the difficulties of a simplistic language that cannot be extended. But BASIC abounds in the microcomputer world. Almost all of the computers purchased for use in schools have BASIC as their only language. When the student has difficulty with BASIC often a self-fulfilling proficiency is enacted. The teacher expects difficulty because computers are mathematical and therefore difficult.

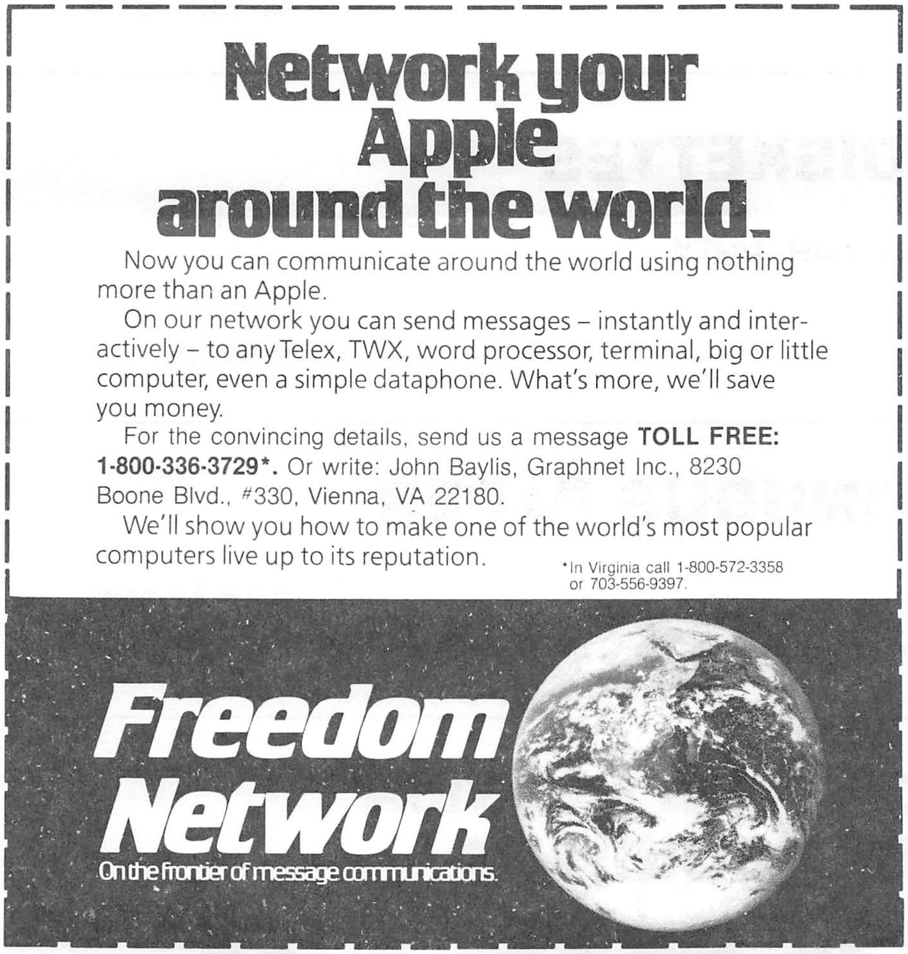
At a seminar for parents and teachers of gifted and talented children, Dr. Robert Taylor, a noted consultant on gifted programs, stated that the Department of Education had presented computer literacy guidelines for seventh graders. It has proposed that in 1983 graduating seventh graders be clasified deficient in math and language skills if they do not have computer experience. By 1986 similarly deprived seventh graders will be clasified as math and language illiterate. If the Department of Education is going to mandate computers in the schools then perhaps design engineers shouldn't be the sole determiners of what a computer can and can't do. We haven't allowed this in other aspects of education so why should technological applications in education be different.

The computer is not a replacement for the instructor nor is it an intelligent task master. Those teachers I have spoken to state their students have the most fun with programs that allow them to explore an unknown environment, and the least enjoyment from programs that drill. If a student's reward for doing well on a math test is simply doing more summations, it is hardly a reward at all. The educational dictum is: CONTENT ==> PROCESS ==> PRODUCT. The computer is not the product of a good educational environment but a part of the process. Do we want computer programmed children or children that program computers?

In Mindstorms, Papert concerns himself with the proposition that all children are explorers of their world, learning naturally and building models. As they continue to learn, they assimilate models from their environment into the previous models building newer ones. He proposes that the computer provides a process by which children can build mathematical models even at an early age. Further, he proposes that self identification as being "not mathematical" leads to other self imposed incapabilities. Math is a language that expresses the functioning of the physical world and that to be "not mathematical" is the same simplication that states that a child cannot learn a foreign language. Put a child in France for a summer and then reevaluate the child's foreign language difficulties. In his "Mathland", Papert provides an environment in which children learn to use math as a language to communicate (LOGO is the translator) with the computer which acts as a model builder. Thus children learn to see the computer as a friendly constructor of geometrical shapes and mathematical relations they could only otherwise invent with difficulty.

Last time in LOGO LOCUS I presented a method for introducing extremely young children to LOGO. Subsequently I have learned that there is an ancillary benefit apart from the math environment it provides. My wife expressed her concern with our three year old's letter recognition. I have found that creative naming of procedures in my "Easy" derivative of LOGO allows us to introduce alphabet letters. For instance, I had redefined FORWARD to be F. Now I have added S for SQUARE, T for TRIANGLE, and J for JUMP. Together we work on letter recognition which is reinforced by the actions of the LOGO turtle.

Next time, expect more applications with LOGO. I am open to your suggestions and comments. Please address me via the Houston Area Apple Users Group.



## ADVERTISERS' INDEX

| | |
|---|---|
| Applied Eng'g. | 17 |
| John Broderick | 9 |
| CTI | 31 |
| Freedom Network | 30 |
| Re Kramer | 26 |
| Micro-Ware Dist. | 23 |
| Moore Business | 27 |
| Riki Rudolph | 26 |
| Wildcat Computing | 3 |

# APPLE /// PEELINGS

by Mike Kramer

This is the first of what is hoped to be a regular column in the Apple Barrel devoted to hints, tips, techniques, and product reviews for the Apple ///. Although there are just a few ///'s in the club, we will see more and more as time goes by. Starting with the November Saturday meeting, there will be an Apple /// special interest group which I plan to coordinate with the help of Steve Knouse. Initially it is expected that the meetings will consist of reviews of software products for the Apple ///. As experience is gained, there will be tutorials in Apple /// Business BASIC, CP/M, and Pascal. On with some tips.

## AVOIDING SYNTAX ERRORS IN BUSINESS BASIC

Apple /// Business BASIC is somewhat less forgiving with entry of BASIC program code than Applesoft is. For example, you must type "GOTO 100" rather than "GOTO100". It is possible to type every character correctly and still get the dreaded SYNTAX ERROR. Although this may seem to be a step backward, it permits the programmer in most cases to use reserved words in variable names. No longer is INPUT$ an invalid name because of the reserved word IN. Apple wrote the Business BASIC interpreter so that it lists the reserved words in upper case regardless of how it was typed in. The trick is therefore to type everything in lower case so that the BASIC reserved words will stand out when the program is listed. Then if you get a syntax error you can determine very quickly if it is due to spelling errors, spacing problems, etc.

## LISTING ON THE PRINTER IN BUSINESS BASIC

One of the things you lose with the Apple /// is the "PR#n" command to list a program or disk catalog on the printer. You also lose the problem of how to print output wider than 40 columns or how to read a file a record at a time and print the records to the printer without repeatedly closing the disk file. It is not immediately obvious to the new Apple /// user, however, how to get his listing or catalog on a printer. First, it is important to understand that the Apple /// treats output devices as though they were files, permitting up to 10 files and/or devices to be activated at one time. To list a program or file enter the following commands:

```
) OPEN #m AS OUTPUT,pathname
) OUTPUT #m
) LIST
```

Where "m" is an integer from 1..10 and "pathname" is the name of the output device driver such as .PRINTER, .SILENTYPE, or .CONSOLE. To direct utput to a file the pathname would be of the form .Dn/filename, where n is the number of the disk drive (1..4). To catalog the disk just substitute CAT or CATALOG. To end output to a specific device type CLOSE #m or CLOSE to close all devices.

As mentioned before, it is possible with Apple /// BASIC to have as many as ten devices or files active simultaneously, permitting a disk file to be read and dumped to the screen, a printer, and another file. If you have tried this in Applesoft you would really appreciate Apple /// Business BASIC.

## SOFTALK APPLE /// BUSINESS BASIC TUTORIAL

The manuals provided with Business BASIC are very good as are most of Apple's manuals. They are references rather than tutorials so many of the features available are not obvious. The best (and only) Apple /// Business BASIC tutorial available is SOFTALK Magazine's ongoing series by Taylor Pohlman entitled "The Third BASIC". Mr. Pohlman assumes the reader has a working knowledge of BASIC and dives right into fairly involved example programs, providing good explanations of how they work. They examples provide a good collection of useful programs including HEX-ASCII file dumps, bi-directional scrolling through text files, etc. If you are getting into Apple /// Business BASIC this series is a must. There is no indication that the articles will be offered in book form as SOFTALK did with Roger Wagner's Assembly Lines Series, but if there is enough demand, who knows.

## ALTERNATIVES TO APPLE WRITER ///

Many of you have been using Apple Writer /// because little else was available in the way of Word Processors for the Apple /// (Word Juggler is also available for the /// but I've never seen it). The introduction of the Apple /// CP/M Softcard has made it possible to run both Word Star and Palantir word processors on the Apple ///. Aka DeMesa has Wordstar running on his machine and the folks at Designer Software have made the necessary changes to get Palantir running (so what did you think this was being typed on?). One of the nice features of Apple's implementation of CP/M on the /// is disk compatibility between Apple ][ and /// versions. This permits reading a disk file on either machine without translation from one format to another. Apple Writer does provide a Utility Disk which will convert Apple Writer /// files to Apple Writer ][ files and vice versa, but the procedure is somewhat involved and it runs only on the ///.

## CONVERSION OF SOS TEXT TO DOS AND VICE VERSA

The Apple Writer /// utility is very useful in converting text files, including those produced by Visicalc, back and forth from the Apple ][ and ///. Many programs (PFS, Visicalc) provide capability to convert files from ][ to /// format, but none to my knowledge permit conversion the other way. Unfortunately the Apple Writer utility does not work on Apple /// Pascal or BASIC data files, which is what PFS, Quickfile, and many other programs produce.

---===*===---

## HIGHLIGHTS OF HAAUG MEETING, October 14, 1982

Robin Cox was appointed Chairman of the Nominating Committee. Elections for new officers will be held at the Thursday night meeting in December. Volunteers were requested to serve with Robin on the Committee.

The date of the next Saturday meeting falls in the middle of Apple Fest, which will be held in Houston at the Albert Thomas Convention Center October 28-31. The Saturdav meetings will be held but no activites are planned. Admission to the Apple Fest is $5 for adults and $3 for children. It is being held in conjunction with the Gulf Coast Computer Show.

Membership renewals should be sent to Lee Gilbreth and not to Mike Kramer. Sending them to Mike will cause a delay which may cause you not to receive the current copy of the newsletter.

Newsletter article response has been great. Advertising has fallen off, and we need the advertising to pay for the printing of the newsletter. The club has decided that "For Sale" announcements at club meetings are inappropriate, but anyone having something to sell mav put a sign up in the meeting room.

"Call A.P.P.L.E" has a bulletin board on Source. Members should see page 77 in the current issue of "Call A.P.P.L.E" for full details.

Approximately 20 members of the club have been attempting to bring the club software library up to date. They are requested to have their final versions at the November meeting.

Jon Stevens presented the program at the meeting entitled "Why Pascal?"

---===*===---

## HAAUG SUPPORTS HOUSTON PUBLIC LIBRARY

In response to a request by the downtown branch of the Houston Public Library, HAAUG has sponsored subscriptions to Peelings II, Nibble, and Softalk and will donate the exchange newsletters from Washington Apple Pi, Call-APPLE, Apple Orchard, Northern Illinois Apple Users Group, and most of the major Apple users groups from around the country as they are received. Since most of you did not know about the exchange newsletters (which are obtained as sources of articles for the Apple Barrel), this will give each of you an opportunity to benefit from these publications.

## CREATIVE EDUCATION - Who says learning can't be fun?

Re Kramer

"Tic Tac Show" & "The Game Show" - Price $4Ø - Computer Advanced Ideas

Unfortunately, most of the educational software I've seen has been written for computer and education-oriented kids - in other words, not the kids who really need a creative approach to learning. A lot of the programs are sound educational devices - with emphasis on repetition and learning objectives - however, as most parents know, motivation is 99% of the game. Getting the "potential student" to sit down to learn something is the greatest obstacle.

In the last few weeks I've had the opportunity to review some software which takes a very creative approach to the idea of using a computer in education - and it's fun! I can tell by the way the kids - and adults - react to the programs when they see them in operation. "Tic Tac Show" and "The Game Show". are both designed after TV game shows and are delightful. The graphics game commentator turns to each team participant to ask the questions, and the participants react accordingly, including throwing their hands in the air and cheering when points are scored for the individual teams.

"Tic Tac Show" asks the questions and then gives you the answer if you miss the question. "The Game Show" continues with a series of questions with decreasing points for the team according to the number of related questions on the subject. Subject diskettes loaded with questions are available, but creation of games with user-designed questions is really easy. So far, I have not had to refer to the manual. These are really good! Look for them at your local computer store

---===*===---

## SUPPORTING STORES

The following stores support H.A.A.U.G. as indicated. Be sure to show your appreciation by patronizing them.

Computer City, 127Ø4 North Freeway, Houston, TX, 713-821-27Ø2
1Ø% Discount to H.A.A.U.G. members.

Moore Business Center, 112Ø Smith, Houston, TX, 713-237-9Ø63
1Ø% Discount to H.A.A.U.G. members.
Prints Apple Barrel at cost.

Micro Solutions, 9949 Harwin #E, Houston, TX, 713-789-5443
1Ø% Disco'int to H.A.A.U.G. members.

## SATURDAY SESSION

### HAAUG ROOM ASSIGNMENTS

| TIME | RM2Ø4 | MAIN | RM2Ø8 | RM228 |
|---|---|---|---|---|
| NOON | CP/M_____ | BASIC___ | PASCAL__ | EDUC.___ |
| 123Ø | CP/M_____ | BASIC___ | PASCAL__ | EDUC.___ |
| 1:ØØ | CP/M_____ | BASIC___ | PASCAL__ | EDUC.___ |
| 1:3Ø | NEW MEMB | BASIC___ | PASCAL__ | EDUC.___ |
| 2:ØØ | GEN MEET | __________ | __________ | __________ |
| 2:3Ø | SPECIAL_ | SOFTWARE | ADVANCED | BUSINESS |
| 3:ØØ | SPECIAL_ | SOFTWARE | ADVANCED | BUSINESS |
| 3:3Ø | GAME SIG | SOFTWARE | ADVANCED | BUSINESS |
| 4:ØØ | GAME SIG | SOFTWARE | ADVANCED | __________ |
| 4:3Ø | GAME SIG | SOFTWARE | STAT SIG | __________ |
| 5:ØØ | __________ | SOFTWARE | STAT SIG | __________ |
| 5:3Ø | __________ | SOFTWARE | STAT SIG | __________ |
| 6:ØØ | __________ | SOFTWARE | STAT SIG | __________ |

SIG CHAIRMEN:
CALL GUS FOR RESERVATIONS 481-5329

---===*===---

### SPECIAL INTEREST GROUPS

Members who share interests are encouraged to join or form Special Interest Groups to more fully explore their fields. These groups meet separately from the regular meetings at times convenient for the members. If you would like to become involved in a special interest group, either call the HOTLINE. Lists of members with specific interests can be generated on request from the HAAUG MEMBERSHIP SURVEY data base.

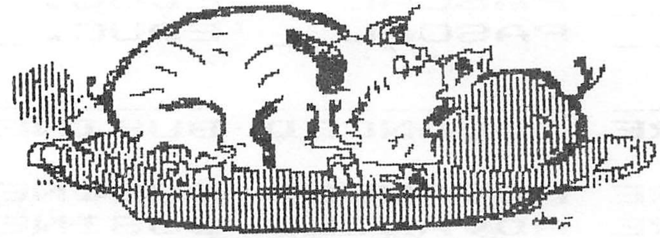
### SIG CHAIRMEN

| | | | |
|---|---|---|---|
| Business | Rudge Allen | CP/M | Jim Huck |
| Pascal | Jon Stevens | Sci/Engg | Mike Conway |
| Statistics | Lindsay Reed | Stocks | George Marsden |
| Education | Fred Yates | FORTH | Steve Knouse |
| Assembler | Robin Cox | BASIC | Gus Gusmorino |
| DOS | Bill Zahrt | Games | Bill Muhlhausen |
| Adv.Topics | Tom Murdock | Apple /// | Mike Kramer |

---===*===---

Houston Area Apple Users Group
APPLE BARREL
2218 Running Springs
Kingwood, TX 77339

VOLUME 5 NO. 8 OCTOBER/NOVEMBER 1982



H.A.A.U.G.