

Bigger and Better...and More!

Editorial by Alan Sheltra

The MOTD now has a new look and a new schedule. It also has a new editor, Alan Sheltra. That's me. Scott McGee, the former editor, has taken over as the Group's Librarian as he is in a better position to handle the varied OS-9 formats than I was (including OS-9000).

Originally, the MOTD was to be a quarterly newsletter, but it was decided that with news breaking as fast as it does, it would be better to publish with more frequency than before. The MOTD will now be published 6 times a year (bi-monthly) starting with this issue. Better for you, and better to let us tell you what's happening in the World of OS-9.

I'm not new to publishing and have been involved with advertising and graphic design since 1975. Some of you may also recognize my name as the publisher of the "OS9 Underground".

This issue's submissions has us "Awk-ing" and "Bawk-ing", with a tutorial on the OSK "AWK" utility by Zack Sessions and an "AWK-like" utility in ASM by Boisy Pitre (Our President). Boisy also shares a C utility, "Park.c" which will compile under any flavor of OS-9, including OS-9000.

Just a reminder that you can submit your programs or articles for publication in the MOTD. Please send your submissions on a 3.5" or 5.25" format diskette (Coco or any OSK format) to:

Alan Sheltra - MOTD Editor
 OS-9 Users Group
 4650 Cahuenga Blvd. Ste. #7
 Toluca Lake, Ca. 91602

You can also e-mail your submission direct to "zog!motd@abode.ttank.com", either case gets your submission direct to me.

Next issue should have a report (and pix) of the Atlanta Coco Fest. The OS-9 Users Group will be there, come and say "Hello" to the guys!

Until next time...!

-Alan Sheltra - MOTD Editor

An Introduction to AWK by Zack C. Sessions

In the February 1992 issue of "68xxx Machines" (Now the OS9 Underground), in his column Bob van der Poel talks about solving complex problems with a few standard commands along with one which is readily available. I was pleasantly surprised to see that Bob was talking about GAWK. I recently was required to get more "intimate" with AWK for a college course, and had already put some work in this artical since I, too had gotten a copy of GAWK for my MM/1. So, after reading Bob's article, I thought I'd finish this article.

First, some history. The AWK programming language first came into being in 1977, being the brainchild of Alfred V. Aho, Peter Weinberger, and Brian W. Kernighan, all of Bell Labs. Knowing this, it is now obvious how the authors thought of AWK as the name of the language. [Hint: Look at their last name's only!] The authors saw the power, yet shortcomings of the UNIX utilities **grep**, **fgrep** and **sed**. They adopted a goal of developing a pattern-scanning language that would understand fields, one with patterns to match field and action to manipulate them.

The current version of AWK was released in 1985, and is documented in a book by Aho, Kernighan and Weinberger called "The AWK Programming Language", ISBN 0-201-0798-X. This version as implemented on most UNIX systems is called "nawk" for "new awk", and the original 1977 version of AWK remains just "awk".

The Free Software Foundation, as expected, released their version of AWK called GNU AWK, or GAWK in 1986. The most recent version, V2.11 was released only recently. It supposedly supports all the features and functions of AWK as descibed in the aforementioned book. (I have found a few situations where GAWK does not fully function as descibed in the book, but nothing major.) This version of GAWK is available for OSK from several sources. I can send you a copy for the cost of the disk and mailing charges. Write to me if interested. (P.O. Box 540, Castle Hayne, NC 28429)

Okay, do I have your undivided attention? I now will address the central and most impotant issue of our discussion on AWK. Just what the heck is it, what does it do and what good is it to me? Here on out, I will refer to AWK as GAWK, since all example commands are depicted how I have used them on my MM/1. But essentially, GAWK is AWK. I will be using the dollar sign (\$) to indicate the SHELL prompt in the example commands.

Directory of MOTD_EARLY_FALL :

Editor's Notes (Editorial)

by Alan Sheltra Page One

Introduction to AWK (Tutorial)

by Zack Sessions Page One

From the Desk of Boisy (Editorial)

by Boisy G. Pitre Page Two

Park those Drives! (Program)

by Scott McGee and Boisy G. Pitre Page Five

Nothing to BAWK at. - Part I (Program)

by Boisy G. Pitre Page Six

From the Desk of Boisy... (Editorial)

User Group President Boisy G. Pitre

OS-9000: Power to the Masses

Most, if not all OS-9 users have heard of OS-9000. In short, it is an enhanced version of OS-9/68K with some major differences:

- ∞ OS-9000 is written in C, whereas OS-9/68K is written in 68000 assemble language

- ∞ Because OS-9000 is written in C, it is portable to a number of hardware platforms, including CISC and RISC processors.

- ∞ OS-9000 is available for 68010 and higher processors as well as 386/486 computers.

Prices of Intel-based computers have dropped rapidly in the past few years. With the introduction of the Intel 586, prices on the 386 and 486 are dropping even more. From a standpoint of cost, OS-9000 represents a tremendous value: a powerful operating system for real-time and development applications which runs on widely-available and relatively inexpensive hardware.

Microware's recent release of OS-9000 version 1.3 includes the Microware C Compiler and VPC (Virtual PC) software. VPC allows one to run DOS applications under OS-9000 while maintaining full multi-tasking capabilities. And, OS-9000 comes with an easy to use installation program which takes care of partitioning your hard drive device and installing the OS-9000 system.

Personally, I am surprised that OS-9 hardware vendors have not taken advantage of the tremendous market potential here. Imagine if you will, a complete 386 computer system bundled with OS-9000! An excellent sales opportunity exists. Not only could such an investment be a financial success, it would give OS-9 users a base on which to grow and expand.

OS-9000 is the perfect example of a state-of-the-art operating system that can adapt to the constant change of the computer industry. The OS-9 Users Group is committed to the establishing support for OS-9000 and VPC. It's an important commitment.

by Boisy G. Pitre

GAWK is a very intelligent file processor. It can process several files with a single invocation. A typical GAWK command would look like:

```
$ gawk 'awk program' [file(s)]
```

If no files are listed, GAWK read from the Standard Input Path, making it perfect for receiving its data from a pipe, as you will see later. If more than one file is listed, all files are processed, but only one at a time, from left to right.

If the awk program is a multi-line program, or if you plan to use it often, you would want to create the awk program in a separate file and invoke the gawk command:

```
$ gawk -f awk.program {file(s)}
```

Where awk.program is the filename of the awk program file.

The key part of either format is the "awk program". That is where all of the intelligence is. An awk program is one or more of awk program statements. Each statement has two parts, a pattern match, and an action. Both parts are optional, but one has to be there. So, the structure of the awk program would be:

```
[pattern] [{action}]
```

The square brackets indicate each part is optional, and actually, if omitted there is a default value for each of them. The action portion of an awk program statement is distinguished by being surrounded by curly braces. Each awk program is applied to each record in the ASCII files. Here is a simple awk program:

```
{print}
```

This awk program displays each record read to the Standard Output path. Since awk sends its output to the standard output path, it is perfect for piping to a child process. Here there was no pattern, thus the default pattern "all records" was used.

GAWK does some "pre-processing" of the records read from the input stream before passing the data to the awk program, in fact this is one of gawk's most impressive and useful features. Each record is parsed and the "fields" are identified. Each field in the input is a string of characters which does not contain a blank. So the following line of data:

```
Kathy 4.00 10
```

has three fields. The values of these three fields can be used by referencing special variables awk sets up for you. The format for a field variable is \$n where 'n' is the field number. In the above record, \$1 would contain the character string "Kathy", \$2 would contain the number 4.00, and \$3 would contain the number 10. Yes, awk determines if the field is alphanumeric or numeric. There is another special variable name you can use to reference the input line. Variable \$0 refers to the entire line. In this record, variable \$0 would contain the character string, "Kathy 4.00 10". The default action for an awk program is equivalent to {Kathy 4.00 10}.

The "n" portion of the field variable can be variable itself. For example, the following awk program will print each field of each record in a separate line.

```
{for (i = 1; i < NF; ++i)
  print $i
}
```

Note again that the field's variable names start with 1. \$1 is the first field, \$0 is not. \$0 is the ENTIRE current input record, I'll talk about what NF is in just a second. You might also notice the similarity of awk action statements to C programming statements.

There is also a few other special built-in variables. NR contains the number of the current input record. Since an END pattern's action is executed after all input data has been processed, in an end pattern's action the built-in variable NR contains the number of records read from the input.

NF is the number of fields in the current record. So, the variable \$NF would represent the LAST field in the current input record. There are several other built-in variables, all of which are beyond the scope of this article. The awk programmer can also use user defined variables simply by referencing them, as I used the variable "i" in the action example above. The data type, string or numeric, is determined by the context. Arrays of numbers and strings are even supported.

All user defined variables also are automatically given an initial value the very first time they are referenced in an awk program statement. The initial value for numeric variables is 0, and for string variables "". Datatype is interpreted from the context it is used in.

The scope of variables is also important to realize, but is also beyond the scope of this article. (No pun intended!) For now, consider field variables as "global", known to all patterns and actions, and variables referenced in an action are local to that action.

The pattern portion is a little more difficult to fully understand especially the regular expressions, unless you are familiar with the concepts of regular expressions a la UNIX. The pattern may be any one of the following:

- 1) The string BEGIN.
- 2) The string END.
- 3) An expression.
- 4) A regular expression.
- 5) A compound pattern.
- 6) A pattern.

I'm not going into the last two at all, and the fourth one only briefly. I will devote an entire article on the last three types of patterns.

BEGIN and END are special patterns. They indicate that their associated actions (it is of little use to have a BEGIN and END with no action!), are only performed at special times. The BEGIN action is always processed BEFORE ANY data has been read. The END action is ALWAYS processed AFTER ALL data has been processed.

A pattern which is in the format of an expression is a comparison between two expressions. All standard comparison operators used in the C programming language are recognized, !=, ==, >=, >, plus two others, > means "is matched by" and != means "is not matched by". I'll talk about these last two later. A typical expression pattern would be:

\$2 >= \$3

The pattern would be true if the value of field #2 is less than or equal to the value of field #3. For each record in the input stream that the pattern is true, the pattern's associated action is executed. Since the default action is { print }, if the above pattern were the entire awk program, then for each record which the pattern were true, the entire record would be written to standard output. Since expressions are allowed in the two items to compare, the following is a valid expression pattern:

\$1 / 2 >= \$4 * \$5

In this case, field 1 is divided by a constant 2. That value is compared to the product of the values of fields 4 and 5, and the pattern is true if it is less than or equal.

A pattern can also be what is called a "string matching pattern". In most cases, this is usually in the form of a single regular expression. To signify that a pattern is a regular expression, it must be enclosed in slashes. Here's an example:

/Mary/

In this case, the pattern is true for any record which contains the substring "Mary", and thus the pattern's associated action would be performed.

Now, on to some actions! An action is one or more valid awk action statements. These look much like C programming statements, and indeed, some are identical and function the same. Perhaps the most common action of the awk action clauses is to output something with either a print or a printf function. "printf" works exactly like the C function does. The print statement command:

print("%s %s %s/n",\$1,\$2,\$3)

The fields are automatically separated by a space. This example assumes that the three fields are all strings. Awk is smart enough to know the difference. Note that the print also does an implied new line at the end of its data. Awk program actions can also "if" (with optional "else"), "while", "for", "do while", "break", "continue", "next" and "exit" statements.

I'll go deeper in subsequent articles. Now at risk of making Bob mad at me, I'm going to analyze his awk programming skills using his February, 1992 article as a guide. His first awk program is a simple one:

\$1 < /bsr/ { print \$2 }

Well, nothing I can say about that. Short, sweet and fuctional. But looking ahead I see that this awk program is intended to be run on several different files and since it's short, it is being run supplied on the command line itself. In a UNIX environment, that is fine, but on my MM/1 using GAWK, there is something to consider. Each time you run a command like:

```
$ gawk '$0 < /Asia/ { print $3, $4 * $5 }' countries
```

there is a file created in the /dd/TMP directory which contains the awk program. Each time it is run, a SEPARATE file is created, even if the awk program being executed over and over and over is exactly the same program. So, every so often, you need to clear out the /DD/TMP directory. So, in Bob's procedure, for this reason, I would have extracted the awk program for both the gawk commands to go out into an external file and use the -f option.

Next program, I got a few observations. Bob uses a BEGIN pattern merely to initialize a variable to 0. Since all variables are given an initial value the very first time they are ever referenced, and if they are used in the context of numerical expression, they assume the value of 0. So, the BEGIN pattern and it's action are redundant and not needed. I can't reall improve on the rest of the second awk program.

I will finish up with an awk success story which strengthens Bob's premise in his article, that is to use the tools you have. I have just downloaded several files and I wanted to set the attributes of the files to public read/write. Ththese files were the only files in the directory which were created on that date, but there were other file in that same directory I didn't want to mess with. So consider the following command:

```
$ dir -e ! gawk '$2 == "92/03/11" { print $7 }' ! attr -z -pr -pw
```

Actually, that is not how I first did it. My first attempt was even cruder:

```
$ dir -e ! grep 92/03/11 ! gawk '{ print $7 }' ! attr -z -pr -pw
```

Actually, if grep is already in memory, the second version would probably run faster, um, nope, I was thinking we would save a lot by having grep do the pattern matching, but gawk will still parse the entire record before even processing the pattern, in this case, process all records. But, the overhead of creating the fourth process needs to be considered also. It was pretty to watch in a procs display in another window!

Next time, more on the complex pattern types and regular expressions.

Zack C. Seesions

AWESOME BOOTFILE EDITOR! KWIKGEN v1.01

Still using OS9Gen, Cobbler, or Config?
Get a real bootfile editor!

EzGen v1.09 vs. KwikGen v1.01
5 minutes 40 sec. 44 SECONDS!

* Identical operations performed on identical fragmented boot disks
- 2 deletes and one insert performed by both utilities

- Editing done in memory
- Load boot from disk or memory
- Patch modules
- Change order of modules in seconds
- 100% assembly code
- Make multiple boot disks in one session
- Edit existing boot files in place easily
- Load kernel from disk or mem and write to disk

KwikGen requires OS9 Level I, or II.

\$24.95

KWIKZAP v1.1

- display updating is instantaneous
- 'smart' verify command
- work on file or stack
- searching functions
- 100% assembly code
- configurable environment
- dynamic sector stack
- allows editing of nibbles or half bytes
- built in help - easy to use

KwikZap requires OS9 Level II.

\$24.95

Experience GALE FORCE speed!

Shipping and handling is \$4.00.
Call or write for our free catalogue
Please call for Canadian prices



Checks: Allow 4 - 6 weeks for delivery.
Money orders: processed immediately for KWIK delivery

Send check or money order to:
Gale Force Enterprises
P.O. Box 6806 Station T, Vancouver,
B.C., Canada V6N 5L4

(604) 271-1655
B AM - 5 PM PST (voice)
SPM - B AM (support BBS)

AniMajik Productions

NEW!
CLOUD_09 - by Albert P. Marsh

The BEST Graphics/Animation Editor for the CoCo! Tools include: Line, Box, Ellipse, Fill, Pencil, Brush, Flow, Spray, Text, FatBlox, Palette. Work with up to 8 animation pages. Copy one page to another. Complete control of animation speed. Edit/Save/Load VEF picture files.

Req: OS9 Level 2, Multi-Vue and 512K 34.95

TShell V3.13.02 - by Paul Pollock

A revolutionary New Program... "TShell" does most of what Multi-Vue does at up to 5 times the speed! TShell will run most programs with one keypress and use standard MV AIF files. Delete, Copy, Rename files all with 1 or 2 keystrokes! Many utilities included.

WINDINT and Multi-Vue NOT required!
Req: OS9 Level 2 512K

TShell (with printed manual) \$39.95

TShell (with "printer-ready" manual, TDoc included to make the job easy!)
You print it... you save! \$29.95

SunDialer "WarGames" Dialer - by John Powers
(Includes versions for ACIA and SACA)
Req: OS9 Level 2 and 512K \$19.95

DCom - Basic09 Decompiler - by Wayne Campbell
Req: OS9 Level 2, 512K \$24.95

Coming Soon!

TAKENOTE
COCO TYCOON V2.0
COCODEX
MEMMATCH
SIG Net V4

(Prices Subject to Change without Notice)

Send Checks or M.O.'s
4650 Cahuenga Blvd. Ste #7
Toluca Lake, Ca. 91602

(818)
761-4135

Park Those Drives!

by Scott McGee & Boisy G. Pitre

Park, a utility by Scott McGee and Boisy G. Pitre for parking not only your hard drives, but your Disk and Tape drives as well.

The C listing below should be able to compile on the 6809 C Compiler or those used on os9-68K and OS-9000.

Happy Parking!

```
/*
 * Park - Resets an RBF device's read/write heads .
 *
 * This utility uses the _ss_rest() call to restore an RBF device's
 * read/write head to track 0. Park works on most RBF devices,
 * including floppy disk drives. For SBF devices, park rewinds the
 * tape.
 *
 * For 6809 C compilers, the Kreider library must be linked to
 * resolve the _ss_rest() function.
 *
 * Written by: Scott McGee & Boisy G. Pitre
 * (C) 1992 - The OS-9 Users Group
 */

extern int errno;

#include <stdio.h>
#include <errno.h>
#include <modes.h>
#ifdef _OS9000
#include <sg_codes.h>
#else
#include <sgstat.h>
#endif

#ifndef TRUE
#define TRUE 1
#define FALSE 0
#endif

main(argc, argv)
int argc;
char *argv[];
{
    int quiet = FALSE;

    if (argc == 1) ShowHelp();
    while (argv[1]) {
        if (argv[1][0] == '-')
            switch (toupper(argv[1][1])) {
                case 'Q' : quiet = TRUE;
                    break;
                default : ShowHelp();
            }
        else ParkDev(argv[1], quiet);
        ++argv;
    }
}
```

```

ParkDev(device, quiet)
char *device;
int quiet;
{
    int path;
    char namebuff[32];

    strcpy(namebuff, device);
    if ((path = open(strcat(namebuff, "@"), 1)) == -1) {
        fprintf(stderr, "Can't open %s\n", device);
        exit(errno);
    }
    if (_ss_rest(path) == -1) {
        fprintf(stderr, "Can't park %s\n", device);
        exit(errno);
    }
    if (!quiet) printf("%s parked...\n", device);
    close(path);
}

ShowHelp()
{
    fprintf(stderr, "Park - RBF device parking utility\n");
    fprintf(stderr, "      (C) 1992 OS-9 Users Group\n");
    fprintf(stderr, "  Usage:  Park [-q?] [/dev] [...] \n");
    fprintf(stderr, "          -q = Quiet - no output\n");
    fprintf(stderr, "          -? = Show this help message\n");
    exit(0);
}

```

Nothing to BAWK at. - Part One

by Boisy G. Pitre

[Editor's Note: Due to space limitations, BAWK.ASM will continue in the next issue of the MOTD.]

```

*****
* BAWK Edition 3 - (Boisy's Awk) - Line processing utility
*
* (C) 1992 Boisy G. Pitre
*
* BAWK is a line processing utility "similar", but not exactly the
* same as the UNIX counterpart. For one, this version of BAWK is NOT
* a processing language. Its a line processing program. Don't get
* BAWK confused with AWK. They are two totally different programs.
*
* BAWK takes advantages of "fields" in lines of text. A field is a
* word or symbol, separated by whitespace. Each field is numbered
* sequentially from left to right. An example of how BAWK see's
* fields in lines of text:
*
*      1 2 3 4
*      my name is boisy
*
* The numbers above the words in the line represent the field
* numbers. BAWK can extract any or all of these fields by their
* field number in any order, for a wide variety of useful formats.
* The formats can then be piped to another program or to a file
* for processing.
*
* Usage: BAWK [-a# -i -d? -l -f -F] "format field" [file] [...]
*
* NOTICE: Options are CASE SENSITIVE!
*          Be aware of this when using BAWK.
*
* Opts:
* -d? = the optional delimiter you wish to use.
*       Ex. to use a colon as a delimiter in addition to the
*       already used space, use -d:
*
* -a# = Anchor start field to #th column.
*
* -i = prints lines even if the field doesn't exist.
*       This option tells BAWK to print the format string
*       even if the specified field is not found on the
*       line. Default is OFF.
*
* -l = Prints the name of the file currently being scanned.

```

```

*
* -f = Forks a shell with the expanded line as a parameter.
*       The expanded line buffer is NOT printed.
*
* -F = Forks a shell with the expanded line as a parameter,
*       and prints the expanded line to StdOut.
*
* The format field can contain any characters you wish. The
* special * character '$' is used to denote field positions.
* (Fields are from 1 to 255).
* If you wanted to extract the 5th field in a line, you would
* include $5 in the format string. BAWK would interpret this to
* be the fifth field, and would then expand it appropriately.
* Looking at the previous example:
*
*      1 2 3 4
*      my name is boisy
*
* NOTE: there is a special field, $0, which denotes the ENTIRE
* input line.
*
* To tell BAWK to print the 3rd field in that line, the
* following command line would work:
*
*      echo my name is boisy | bawk "$3"
*
* Note that the format string is ALWAYS contained in '"' quotes.
* You may also include non-specific info as part of the field.
* I use the following line in my /DD/SYS/CRONTAB file to stop my
* BBS at 7:00 each morning:
*
*      procs | fgrep -i "tsmon" | bawk -f "kill $1"
*
* This line uses PROCS' output, and pipes it into FGREP. FGREP
* then throws away any lines that don't contain the word tsmon.
* The remaining lines that do contain the keyword are then piped
* to BAWK. BAWK takes the first field of that line (the Process
* ID number of tsmon), and combines it in a format whose output
* looks like: kill 3 (assuming 3 is the ID of the tsmon process).
* Finally, that line is used as a parameter for shell execution.
* You may also specify a filename or a list of them AFTER the
* format string. When using filenames, you can tell BAWK to print
* the name of the file it is currently scanning (to StdOut) using
* the -f option.
* To use '$' as a regular character in a format string, use $$
* BAWK does not interpret this as the '$' field specifier.
*
* You can mix and match any number of fields for interesting
* combinations. If you prefer military time instead of DATE's
* regular output, try this:
*
*      date | bawk -d, "$2 $1 $3"

```

```

* This example also introduced the -d option. This option allows
* you to specify another delimiter in addition to the already used
* space character. A good example of this would be finding the 5th
* field in the /DD/SYS/PASSWORD file, whose delimiter is a comma,
* NOT a space.
* Another useful option is -i. This allows the inclusion of
* printing the format string even though that field is not found on
* the line. Take the previous example again:
*
* 1 2 3 4
* my name is boisy
*
* The following command line would print nothing:
* echo my name is boisy | bawk "Name $5"
*
* BUT this line will output 'Name ':
* echo my name is boisy | bawk -i "Name $5"
*
* Keep in mind that if the -i option is not used, NO lines will
* be printed unless ALL fields are qualified.
*
* One other option worth noting is the -a option. This option
* sets an "anchor" to the #th column. Using our infamous example:
*
* 1 2 3 4
* my name is boisy
*
* The following command line would print 'name' because the
* anchor is set to the 3rd column (which is a space, ignored by BAWK):
* echo my name is boisy | bawk -a3 "$1"
*
* To print the names of all the modules in block $3F using MDIR:
* mdir e | grep "3F" | bawk "Module: $8"
*
* BAWK has very useful applications, making it a worthy addition to
* your OS-9 toolbox. In addition to these features, BAWK works GREAT
* with Shell's wildcards!
*
* By: Boisy G. Pitre
* Southern Station, Box 8455
* Hattiesburg, MS 39406-8455
* Internet: bgpitre@seabass.st.usm.edu

```

```

nam      BAWK
ttl      Line processing utility

ifpl
use      /dd/defs/defsfile.dd
endc

mod      Size, Name, Prgrm+Objct, Reent+1, Start, Finish

Name     fcs      /bawk/
Ed       fcb      3              Edition #3

Anchor   rmb      1
Path     rmb      1
IncFlag  rmb      1              Inclusion Flag
FileFlag rmb      1              Show File Flag
ForkFlag rmb      1              Fork Shell Flag
FEFlag   rmb      1              Fork Shell and Echo Flag
Delim     rmb      1              Delimiter storage
FileBuff  rmb      60             Filename buffer
Format    rmb      250            Format buffer
Line      rmb      250            Line buffer
Expline   rmb      4096           Expanded line buffer
Stack     rmb      200
Params    rmb      200
Finish    equ      .

```

```

HelpMess fcc      /Usage: BAWK [-d? -i -l -a#] "format_string" [file] [...]/
fcb      $0d

Shell     fcc      "Shell"
fcb      $0d

FileHead  fdb      $0a0d
fcc      "*** File: "
FileHLen  equ      *-FileHead

```

```

*****
* Subroutines
*****
* Str2Byte - Converts an ASCII string to a single byte
*
* Entry: X - Address of first char in string
*
* Exit:  B - Converted byte
*        X - Last number in string + 1
*

```

```

Str2Byte  clrB
cnvloop   lda      ,x+
          cmpa     #'9
          bhi      cnvdone
          suba     #'0
          blo      cnvdone
          pshs     a
          lda      #10
          mul      addb
          bra      cnvloop
cnvdone   leax     -1,x
          rts

```

```

*****
* Saves filename in buffer and print it
*
* Entry: X - Address where filename is
*
* Exit:  None. File is stored in FileBuff
*

```

```

SaveFile  pshs     x
          leay     FileBuff,u
SaveF2     lda      ,x+
          cmpa     #$20
          bne      SaveF3
          lda      #$0d
SaveF3     sta      ,y+
          cmpa     #$0d
          bne      SaveF2
          puls     x
          rts

```

```

*****
* Sets the anchor
*
* Entry: X - Address of line
*
* Exit:  X - Points to the EOLN char at the
*        end of the line
*

```

```

AncLine    pshs     b              save counter
          tst      Anchor          Anchor to a column other than 1 or 0?
          beq      Return          Nope, process at first column
AncLoop     ldb      Anchor          else move X to anchor point
Anc2        lda      ,x+
          cmpa     #$0d
          beq      BackUp
          decb
          bne      Anc2
BackUp      leax     -1,x
Return      puls     b
          rts

```

```

*****
* Prints filename to StdOut
*
* Entry:  None
*
* Exit:   None
*
* Prints a file header to StdOut along with
* the filename.
*

```

```

PrnFile     pshs     x
          leax     FileHead,pcr
          lda      #1
          ldy      #FileHLen
          os9      I$Write
          lbcs     Error
          leax     FileBuff,u
          lda      #1
          ldy      #60
          os9      I$WritLn
          lbcs     Error
          puls     x
          rts

```

(Continued Next Issue)

See us at the Atlanta CoCo Fest!

October 3rd & 4th

SYSTEM IV

The 68000 Computer serving customers world-wide

This high-quality, high performance 68000 computer was designed for and is accepted by industry. Perfect low-cost work-station, development platform or fun machine. Powerful, flexible and expandable inexpensively. Run MS-DOS software with the optional ALT-86 card. Supports up to 4 operating systems.

Prices start at \$999.00 with Professional OS-9

For boards and kits, call Peripheral Technology at 404-973-2156

G-WINDOWS

for the SYSTEM IV and PT68K4/2

Multi-tasking - processes continue running when windows are made inactive or are hibernating.

Windows may be re-sized, moved, overlaid, etc.

GUI to start processes by selecting an icon or, start processes from your custom menu or from the command line.

Copy and Paste between windows.

Adds command line editing, command history, and file name expansion.

Runs existing OSK software without modification.

Number of windows and processes limited only by your memory.

Includes GIF viewer.

Includes G-VIEW demo.

G-WINDOWS with DESKTOP \$199.00

G-WINDOWS Developer's Pak \$299.00

Order both for \$449.00

OS-9/68000 SOFTWARE

NEW - DataDex, a free form data management program designed to keep records like a card file system. No programming language to learn. Variable record sizes permitted.

QUICK ED - Screen Editor and Text Formatter

VED ENHANCED - Text Editor

SCULPTOR - Development and Run-Time Systems

FLEXBLINT V4.00 - The C Source Code Checker

WINDOWS - C Source Code Windowing Library

IMP - Intelligent Make Program

CALC-9 - Spreadsheet

VPRINT - Print Formatter (for VED)

M6809 - OS-9 6809 Emulator/Interpreter

DISASM_OS9 - OS-9/68K Disassembler

PROFILE - User State Program Profiler

PAN UTILITIES

delmar co

Middletown Plaza - PO Box 78 - Middletown, DE 19709
302-378-2555 FAX 302-378-2556

Vprint/68000 Text Formatter

The latest addition to our OS-9/68000 product line is the most powerful text formatter available. Vprint will work with any printer from files produced by your favorite editor. Proportional character sets are fully supported as well as most of the special features newer printers have--it even works with laser printers. Standard features include margins settings, indents, headers, footers, etc. Advanced features include multiple column output, repeats, powerful macros with optional parameter passing, internal number registers with many output formats, true footnotes, automatic indexing and table of contents generation, future event testing... And if that's not enough, Vprint has a complete string manipulation language; it supports documentation via change bars, marginal notes and boxed sidebars; and permits i/o redirection to and from pipelines.

Send for a free sample printout demonstrating some of the many advanced features!

Vprint comes with a 100 page manual and loads of sample files. It can be configured by the user to any printer. Vprint costs only \$59.95, plus \$3.00 shipping and handling. To order please send your check or money order and preferred disk format to:

Bob van der Poel Software

PO Box 355
Porthill, ID
USA 83853

PO Box 57
Wynndel, BC
Canada V0B 2N0

Phone 604-866-5772

Advertise in the MOTD Now!
Reasonable Rates!

Contact Alan Sheltra @ (818) 761-4135 for more information

Do your OS9/OSK Machine a favor...

Subscribe to:

The "International" OS9 Underground

Magazine Dedicated to OS9/OSK Users Everywhere!

One Year Subscription (12 Issues) \$18.00

(\$23.00 Canadian, \$27.00 Overseas)

OS-9 User's Group Members Get 2 Extra Issues! (14 in all!)

"The OS9 Underground Magazine"

Fat Cat Publications

4650 Cahuenga Blvd.. Ste #7

Toluca Lake, Ca 91602



ColorSystems

Quality OS-9 Software for
the CoCo3 and the MM/I from IMS

CoCo3 Software

MM/I Software

**Variations of
Solitaire \$34.95**

Variations Included:

Pyramid, Klondike,
Spider, Poker,
and Canfield

**OS-9 Game
Pack \$34.95**

Package Include:

CoCoHello
CoCoYahtzee
KnightsBridge
Minefield
and Sea Battle

**Variations of
Solitaire \$49.95**

Variations Included:

Pyramid, Klondike,
Spider, Poker,
and Canfield

**OS-K Game
Pack \$49.95**

Package Include:

Flip It
Dice Poker
KnightsBridge
Minefield
and Sea Battle

WPSHel \$22.00

Sorry! ColorSystems no longer
carries WPSHel!

Coming Soon for the MM/I:
Super Label Printer
X10 Master Control Program
and Much More!

CoCo and OS-9 Club Members: Have your Club President write to
ColorSystems to ask about our SPECIAL Club Discount Program!

*Official Member of the Interactive Media
Systems Developers Association*

Shipping: FREE for Continental US, \$3.00 for Canada, \$5.00 anywhere else
To order send check (US Bank ONLY) or Money Order (US Funds) to:

ColorSystems

P. O. Box 540

Castle Hayne, NC 28429

For Additional Information call at (919) 675-1706
North Carolina residents please include 4% Sales Tax

Write or call for a FREE copy
of our Catalog!