AUSTRALIAN OS9 NEWSLETTER

EDITOR :
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

(07) 345-5141

MARCH 1989

ATTR Syntax: Attr filename [permissions] Usage : Examine or change the security permissions of a file Opts: -perm = turn off specified permission perm= turn on specified permission -a = inhibi rms : d - directory file s - nc to owner w - write permit to own r - read permit to public pw - e permit to public BACKUP Syntax ge : Copies all data from one de ead error occurs s = single rites BASIC09 Syntax : Basic0 ge BUILD Syntax: Build filenar s from standard input CHD S ge working directory to specifi > Usage: Change execution directory to specified path CMP Syntax: Cmp filename1 filename2 Usage : File comparison utility COBBLER Syntax: Cobbler devname Usage : Creates OS-9 bootstrap file from current boot CONFIG Syntax: Config Usage : Create system boots and system disks COPY Syntax data from one fil E Syntax : Date [t Opts: t = specify ame> Usage : Check directory for wor sters -m = save of unused cluster nly -o = print <devname> }<devn : Del [-x] filenam s : -x = delete x: Deldir directo yntax: Dir [e] [x] the file names x=print executi Usage : Display s converted characters to standard output DSAVE Syntax : Dsave [-opts] [dev] [pathname] Usage : Generates procedure file to copy all files in a directory system Opts : -b make a system disk by using OS9boot if present -b=<path> = make system disk using path ; n = indent for directory levels -d = do not process b makdir command o num K ECHO Syn tandard output ED oriented text edito s text error messages for given error numbers EX Syntax: ex <modname> Usage: Chain to the given module FORMAT Syntax : Format <devname> Usage : Initializes an OS-9 diskette Opts ; R - Ready L - Logical format only "disk name" 1/2 number of sides 'No of

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group

EDITOR : Gordon Bentzen

HELPERS : Bob Devries and Don Berrie

SUPPORT : Brisbane OS9 Level 2 User Group.

Welcome to the March edition of the National OS9 User Group Newsletter. I guess that one of the first things that you will notice, is the new format for the cover page. No, we apologise, it wasn't done using an OS9 machine. For those of you who are interested, it was done on a XT clone using a mouse driven graphics package called VPG, and printed on a Star NX-10 9 pin dot matrix printer in hi-resolution mode. Finally, we had the black and white original photocopied using a red cartridge. We think that it is a quite spectacular result.

We have also decided that it was time for a slight name change, and we hope that you agree that 'Australian OS9 Newsletter' is a better title. The change in title was necessary, because we wanted to ensure that any international readers will be aware of the source of the publication. Already, there has been a subscription enquiry from the USA, and we currently have a subscriber in Nugini. We believe that there are some very talented people within our group, and that they deserve all the recognition that they can get. Consequently, we want to ensure that the source of the material, at least on a geographic basis, is recognized.

So, if you have something to say on the subject of OS9, and want to reach an extremely large geographic area, this is your forum. The articles and suchlike that have been submitted to us from our contributors are greatly appreciated. However, we are constantly looking for further material to include. Remember, this is supposed to be a collective endeavour, and believe me, it gets harder and harder for the current small number of contributors to continue to produce articles which are of continuing interest, on a month by month basis.

This months' issue is once again an interesting collection of OS9 related material.
We had an interesting letter recently from a person who had recently purchased OS9 from Tandy, and had decided to try to modify the system slightly, to suit his own needs. We decided to include his tale of the trials and tribulations of an OS9 beginner, and his pursuit of the mysterious "OS9 BOOT - FAILED" problem. After you read his letter, we hope that those of you who are beginners in the world of OS9, will have a better understanding of the intricacies of the OS9Boot file, and the bootstrapping system. It should also indicate the depth of knowledge available from the so called Tandy technical experts.

Incidentally, for a really good explanation of the bootfile system and the OS9Gen command and process, we recommend that you read the article on page 172 of the November 1988 issue of the US Rainbow magazine entitled "Boot Mysteries Revealed" by Richard. A. White. It is an excellent article.

Also included in this issue of the newsletter, is a precis of the documentation that comes with Shellplus Version 2.0, an extremely powerful replacement for the Shell command.

Bob Devries has included the first installment of the source code for his simple database written in C. This is a very interesting programme as it demonstrates the true versatility of the C language. The source, with virtually no modifications, will compile and run on an MSDOS computer, as well as on a Color Computer! It is also an excellent demonstration of good open programming that will allow this cross compilation.

John Usher, one of our Brisbane members, has kindly supplied the circuit diagram for his RS232 port for the CoCo. If you are interested in trying to make this interface, John will sell you one of his customized printed circuit boards. He also has information about internal mounting for the interface, therefore avoiding the necessity of having a (now discontinued) Multi-Pak.

And, finally, Rosko has submitted another Chapter in his series 'Making the most of Microware C'.

So, if we haven't included something in which you are interested, how about submitting something yourself? Or, if not that, at least tell us where your interests lie, so that perhaps we can try to include something just for you. Until next month, best wishes and happy OS9-ing!

LETTER TO THE EDITOR :-

The OS9 Learner

It is early a.m. and I am taking a break from endless CONFIGS, DSAVES and manual thimbings, not to mention language which would make a bullocky blush. It occurs to me that you may like to hear a learner's trials and tribulations with OS9.

Firstly, about myself. I have been using "greycase" CoCo 1 for five years, programming in Basic using Tandy DOS, modified by Brian Dougan to run 35 track double sided drives. I had some programmes published in "Aust. Rainbow / CoCo" before I severed connections with them, and I belonged to the BrizBiz user group. I have had my eye on OS9 for a few years, but done nothing about it. Now my chance has come; a mate has gone IBM compatible, and offered me his CoCo 3 512k with Tandy FD502 drive, and a completely untouched OS9 Level 2 package. The price is too good to resist and I buy it.

Now my troubles commence. The computer won't work, so off to Tandy with it. Nearly three weeks later it returns - connections shook loose in transit from Northern Territory. I fire up OS9 and the TV screen scrolls like crazy. Phone Tandy - Brisbane can't help, but they pass it to Mt. Druitt (N.S.W.), who ring back in ten minutes. Now, that's good service, and Tandy are to be commended for it. (The last bit of praise they'll get in this article !) "Send the disk to me, and I'll patch it. OS9 was sold in Australia for 60hz U.S.A. power supply".

I have since found out that the disk could have been patched here in Brisbane, and I did at least have the sense to ask that question. "No, must come to Sydney", says Mr. Druitt. The service was quick, and there was no attempt to charge me - fortunately, as you will hear later.

While waiting for the return of my system master disk I enquired of Olaf (Tandy Brisbane), and discovered that a user group exists. At this stage, I was looking for someone to marry my Teac 40 track drive to the Tandy FD502 drive in the one case. I was referred to Bob Devries, and he whipped them together in no time flat. While he was working, I mentioned the foregoing saga, and found out that he could have patched the System Master with no problem. It did come back from Sydney however, and I began sweating over the manual to work out how to switch from 35 track single to 40 track double. (I should mention that I had obtained back copies of this Newsletter form Bob, who, when he found I was interested in subscribing, covered his living-room in one second flat to get an application form.)

CONFIG let me down - I got a "can't link Rel module" error. I tried looking for the cause, and couldn't find it. After a few hours of frustration (and learning - nothing like difficulties to make you learn), I picked up the Newsletter copies to read and, in December's issue, Bob's article on reconfiguring for different drives! But it still didn't work, so back to the manual. I discovered MDIR, and compared the results with the manual. No "Rel" module! And no documentation in the manual about this module (that I could find).

This morning , I swallowed my pride, and revisited Bob Devries. The "Rel" module had been lost in Tandy's patching efforts, and Bob fixed it. And now after many problems this evening (all of my own making), my system is properly configured. Well, not quite all my own making, CONFIG doesn't copy the SYS directory, nor a couple of window files, and I originally DSAVE'd from my 'customised' Master, instead of from the backup of my System Master, and I had to do it all again.

Conclusions:

1. OS9 is a most challenging system, with great apparent flexibility, and I look forward to a lot of fun with it.

2. The manual is good, but it has its deficiencies as mentioned above. Another deficiency is a warning not to use "list" for execution files - use "dump" instead. Try to find any documentaton on "dump" in the manual, even though it is listed in the index!

3. Tandy Mt. Druitt would be well advised to revise their patches for OS9, and joining the National OS9 User Group would be an excellent way to avoid the problems I have had with the System Master disk.

More in future issues, if response dictates.

Ian Clarke.

Well Ian, we do thank you for your "Letter to the Editor" and for relating your early experiences with OS9 as most of us can relate to your frustrations. We have had, and still do have, many frustrations with this powerful operating system, and agree with your conclusion that "OS9 is a challenging system, with great flexibility"

OS9 differs from many operating systems as it can be altered so easily because the user can get to each of the operating system modules and change or replace them, or add others.

Owners of MS-DOS machines for example would know that a new version is often needed when new hardware is added to the system. Want a hard disk? Fine, but version 2.1 of DOS won't work with this. A vers. 3.1 will support a hard drive, so you purchase that. Want access to extended memory? Oh no - better buy vers. 3.21. And on it goes. As a user of DOS you are limited to the use of hardware supported by the version you have, or a version that is available.

OS9 modules can be changed or modified. There is a distinction between "user" and "system" modules. While user modules are loaded and used after bootstrapping a system, system modules are normally included in the Kernel and Boot files. Kernel files contain the programmes that start and manage the system's operation normally in the files OS9p1 and OS9p2. System I/O drivers and descriptors are used by the Kernel to communicate with attached hardware. Normally you cannot change the modules in the Kernel. OS9Boot should contain system modules, and of course user modules should never be there.

Under Microware's Level II, the Kernel includes REL, Boot, OS9p1, OS9p2 & Init. The first three are stored on Track 34 of the boot disk and are loaded when you type DOS to start up OS9. REL resets the system hardware, prepares it for OS9 and calls OS9p1. OS9p1 initializes the system, and Boot loads OS9Boot and the disk I/O managers, drivers & descriptors.

OS9p2, Init, CC3Go, Clock, must be in your OS9Boot file. Init is not a programme. It is a data module containing system constants. OS9p2 handles memory management, the module directory and functions associated with module management, and process control. This is the heart of multitasking capability in OS9. CC3Go now loads "Shell", and "GrfDrv" establishes communications with Shell and starts the 'startup' file.

Shell handles communication between you and the computer. The programme interprets commands that are typed in and calls the proper operating system code to execute them. Though not part of the operating system, it may be considered an application programme that interfaces with the operating system. Shell is designed to be loaded separately from OS9Boot as is GrfDrv. GrfDrv needs to be loaded at the start of an 8k memory block, and therefore must not be merged with any of the other commands.

Well we are now getting into the depths and also off the track. Suffice to say that we can see why Ian had trouble with a boot disk minus REL.

The other important modules to do with hardware communications are "System Managers" (SCF, RBF, PipeMan etc.) and device "Drivers" and "Descriptors".

The standard Drivers supplied with Microware's Level II for the CoCo 3 are CC3IO - handles the terminal functions; CC3Disk - handles floppy disk drives; Piper - works with pipes; RS-232 - driver for T1; etc.

Under level II the Descriptors provide the characteristics of hardware to the appropriate Driver and this makes it easy to modify hardware characteristics. D0, D1, D2 etc. are descriptors for floppy drives, and each of these may provide separate data to the CC3Disk driver.

OS9 Level 1 is not quite as easy, as the Driver CCDisk is hard coded for 35tr. single sided disks and ignores the characteristics in the disk descriptors. In this case the driver needs to be modified or replaced. D.P.Johnson (U.S.A) has available "SDISK" which is a modified device driver which looks for a descriptor and its data in a similar way to Level II There are also many other sources for a modified disk driver for level 1.

This is just one example of how flexible OS9 can be. Just add or change device drivers and descriptors to suit your hardware. JUST! you say?

Regards,
Gordon Bentzen.

Making the Most of Microware C - Part III

by --Rosko!--

NOTE: requires some knowledge of C and assembler (6809 or 68000)

This month I think we'll look at something practical... the development of a (simple) set of functions for your C library, from the design to 6809 and 68000 implementation.

The set of functions (two in all) are to handle string comparisons without attention to upper/lower case, i.e. we want to compare two strings (henceforth S1 and S2) but pretend they're both upper or both lower case. This can be useful when a string is typed in upper or lower case, and is to be compared to a string with mixed case letters.

Step one: How do you compare two strings?

By going through both strings and subtracting a byte in one string from a byte in the other string, until either a difference is found or both strings finish. Remembering that we want to achieve something similar to the standard functions strcmp() and strncmp(), we want to subtract the byte from S2, from the byte from S1. There are three possible outcomes (of interest) from each subtraction:

1. Zero, meaning the bytes are the same.
2. Something less than zero, which means the byte from S2 is bigger than the byte from S1.
3. Something greater than zero, which means the byte from S1 is bigger than the byte from S2.

Once we have found a difference, i.e. the bytes aren't equal, we can stop looking and return the result. We need only return the difference in the two bytes as this tells the caller whether S1 is (alphabetically) smaller than S2, or vice versa.

When the strings are of different lengths, one of the bytes from the strings will be zero (end of string marker), so we can happily return the difference between zero and the other byte.

When the strings are the same, the subtraction will give zero right up to the end so we have to check one of the bytes to see if it is zero already. We then return zero, meaning the strings are the same.

Step two: how do you compare two bytes, ignoring upper/lower case?

We must first convert both bytes to either upper or lower case characters. To do this, we must see first if the byte is alphabetical (we don't want to change digits or punctuation), then convert it to the other case. From upper to lower, we can either add $20 or OR it with $20; from lower to upper either subtract $20 or AND with $df.

When both bytes are of the same case, one can be subtracted from the other.

Step three: implementing all this in assembly language

1. To go through two strings and compare a byte from each, the quickest and easiest way is to have a pointer to each string. We can have one register pointing to our current point in S1, and another for S2. For the 6809, where we normally have only X free for this, we will have to push U on the stack and use it for one of the pointers. For the 68000, we can use A0 and A1.

When we want a byte from one of the strings, we can bump the pointer along to the next byte as we fetch the current byte like

```
LDB ,X+          or          MOVE.B (A0)+,D0
```

2. With one function, we must keep track of the number of bytes compared (to be compatible with strncmp()). On the 6809, this is best done by pushing Y and using U and Y as the pointers, and X as a counter, decrementing it at the bottom of each pass. On the 68000, of course, registers abound, and there is this really neat intruction called DBNE, for decrement and branch if not equal. It first tests for equality, and if so, compares the register you assign to -1, and if it isn't it decrements the register and branches. It was just made for these loops!

3. Since we will be writing two functions which will both need to convert bytes to one case twice (once for S1, once for S2) on each compare, we can code the converting bit into one common routine. We will pass the byte to be converted in B (6809) or D0 (68000).

4. To compare two bytes on the 6809, one must be in an accumulator and one must not. We shall push the converted byte from S2, then subtract it from the converted byte from S1 in B. On the 68000, we can just move the byte from S2 into D1, and subtract D1 from D0.

So, without much further ado, here is the finished result. To demonstrate that it doesn't matter which case you convert alphabetics to for the comparison, I have used an upper-case conversion for 68000 and a lower-case for 6809. Also, to show that setting/reseting the case bit is no different to adding/subtracting the difference, I have gone for setting the bit for lower case in 6809 and subtracting the difference in 68000. (Actually, truth be known I wrote them months apart, needing stricmp() only recently on 68000). And as the 68000 systems generally can 'spare a few bytes' and run 'quick enough anyway', and because I had been passing NULL strings to some of my functions (!), I added a check for NULL strings to the 68000 version.

I hope you find some use for these... and PLEASE tell me if I'm boring!

```
        nam stringsi.a
        ttl Micrwoware C functions for case-ignored string compares
* --Rosko!-- 15/5/88 (3/6/88)
*
* these functions are case-ignorant equivalents of the existing functions
* 'strcmp()' and 'strncmp'. Their use is identical with the exception that
* upper/lower case is irrelevant
*

        psect stringsi_a,0,0,1,0,0

*********
* stricmp (s1, s2) char *s1, *s2;
stricmp: pshs u
        ldu  4,s     get s1
        ldx  6,s     get s2
icmp10  ldb  ,x+
        beq  icmp20  don't get caught on two null's
        bsr  tolower
        pshs b
        ldb  ,u+
        bsr  tolower
        subb ,s+
        bne  icmp30  different so look no further
        bra  icmp10
icmp20  ldb  ,u      .. for s2 ending first
icmp30  sex          return in d
        puls u,pc

* lowercase conversion
tolower  cmpb #'A
        blo  t110
        cmpb #'Z
        bhi  t110
        orb  #$20    make lowercase
t110    rts

**********
* strnicmp (s1, s2, n) char *s1, *s2; int n;
strnicmp: pshs y,u
         ldu  6,s     get s1
         ldy  8,s     get s2
         ldx  10,s    get n
         beq  incmp20 none to do
incmp10  ldb  ,y+
         beq  incmp20 don't get caught on two null's
         bsr  tolower
         pshs b
         ldb  ,u+
```

```
         bsr  tolower
         subb ,s+
         bne  incmp30 different so look no further
         leax -1,x
         bne  incmp10
incmp20  ldb  ,u
incmp30  sex          return in d
         puls y,u,pc

         endsect

* stringsi.a - string compares which ignore upper/lower case
* --Rosko!-- 16/01/89 for the 68000

         psect    stringsi_a,0,0,0,0,0

* int stricmp (s1, s2) char *s1, *s2;
stricmp: tst.l    d0           Check for null
         beq.s    stricmp4
         tst.l    d1           ditto
         beq.s    stricmp4
         movea.l  d0,a0        get S1
         movea.l  d1,a1        get S2
stricmp1 move.b   (a1)+,d0     get byte from S2
         beq.s    stricmp2     ... unless end of string
         bsr.s    d0_upper     mask case
         move.b   d0,d1        and save in D1
         move.b   (a0)+,d0     now get byte from S1
         bsr.s    d0_upper     mask case
         sub.b    d1,d0        get difference
         beq.s    stricmp1     keep going if none
         bra.s    stricmp3     else return it
stricmp2 move.b   (a0)+,d0     .. for when S2 ends first
stricmp3 ext.w    d0
         ext.l    d0
stricmp4 rts

* local toupper routine
d0_upper cmpi.b   #$61,d0   less than 'a'?
         blo.s    d0u_10
         cmpi.b   #$7a,d0   greater than 'z'?
         bhi.s    d0u_10
         subi.b   #$20,d0   subtract difference in cases
d0u_10   rts

* int strnicmp (s1, s2, n) char *s1, *s2; int n;
strnicmp: tst.l   d0           check for NULL
         beq.s    strnicmp4
         tst.l    d1           check for NULL
         beq.s    strnicmp4
         movea.l  d0,a0        get S1
         movea.l  d1,a1        get S2
         move.l   4(sp),d2     get count N
         subq.l   #1,d2        adjust for use in DBNE loop
         bpl.s    strnicmp1    but trap nil or negative count
         moveq.l  #0,d0
         rts
strnicmp1 move.b  (a1)+,d0     get byte from S2
         beq.s    strnicmp2    but runaway if end of string
```

```
        bsr.s   d0_upper     mask case
        move.b  d0,d1        and save in D1
        move.b  (a0)+,d0     get byte from S1
        bsr.s   d0_upper     mask case
        sub.b   d1,d0        get difference
        dbne    d2,strnicmp1 bail out if different or n-- == 0
        bra.s   strnicmp3    go again
strnicmp2 move.b  (a0)+,d0   .. for when S2 ends first
strnicmp3 ext.w   d0
        ext.l   d0
strnicmp4 rts

        ends
```

--Rosko!--

oooooooooOOOOOOOOOOoooooooooo

SHELLPLUS Version 2.0

The following is a precis of the documentation which comes with a newly released version of the CoCo shell module (shellplus), a public domain programme written by Kent Myers, Kevin Darling and Ron Lommardo.

Shellplus will now unload (unlink) the correct module name. Previously, if the module name did not match the command filename you typed, shell would not unload the module, and it would stick in memory until manually unlinked. This shell reads in the actual module name and uses it instead of whatever was on the command line. For example, when a command such as "/d0/cmds/bob" is executed, the module which shell will attempt to unlink will be the module named "bob".

To prevent attempted execution of a write-only device as a procedure file, and to help with use of L-II window startup, this shell checks modes and attempts write of a null to std out when shell starts up.

You can redirect >>> to a write-only device. Before this, the shell would open the path in UPDATE mode.

The quote bug is fixed. That is, leaving off the second quote mark in lines like - format /d0 r "Disk - will not crash the shell.

The EX bug is fixed .. if you type ex with no parameters (to kill a shell) it will not attempt tc fork any module or return an error to the parent process.

Standard error path is now 'un-redirected' after using a pipe.

The shell now runs in what is commonly called no block mode. Under previous versions of shell, if you had a shell running on say T2, and another user was running on Term, there was no convenient way to send messages back and forth without waiting for keyboard input from the receiving shell. In this version, shell puts itself to sleep while waiting for keyboard input. This 'noblock mode' can be turned off using a supplied patch file.

Path= allows you to specify alternate directories to search for commands. If the current execution directory does not contain the desired module, the alternate paths will be searched in the order specified. For example, you could have your commands directory spread across two or more disks.

Settable and useful prompts. Shellplus adds user-settable prompts:

```
p=prompt        The prompt may be up to 21 chars, extra ignored.
                A space, return, or semicolon terminates the prompt.
                Options include:
                    #  - show shell's decimal proc id # (00-99).
                    @  - show device name current std out
```

More than one # or @ is ignored.

Examples:	Prompt Resulting:
p=OS9/@:	OS9/Term:
p=Hi_There!	Hi_There!
p=OS.#.@:	OS.06.T2:
p=	<uses default setup in shell header>

It's easy to start up new shells with custom prompts. Just pass it as a parameter.

To start a shell on say, /W4 with a special prompt, you might type:

```
'shell p=OS9/@?  i=/w4& ' ... results in shell on W4 with prompt 'OS9/W4?'
'shell p="Hello.#[@]" i=/w4& ' ... gives prompt of 'Hello.07[W4]'
```

Shellplus allows global shell scripts to be placed in your execution (CMDS dir). Simply build or copy a shell procedure file to your cmds dir, then set the execution permission bits on it (attr script e pe). This makes it easy to add some commands that you can access almost all the time.

The shell search path becomes: memory, execution dir modules/scripts, data dir scripts.

Uses include printer setup cmds (using display >/p), window commands, and program startups.

For example, you could use a shell script called "fsm" in your commands directory to open a VDG screen on W6, and start Flight Simulator. After exiting from FS, it could reset W6 back to a standard window and everything is done in the background):

```
* FSM - Procedure Command File in CMDS that starts FS-II.
xmode /w6 type=1; display c >/w6
chd /dd/games/fs
(fs <>>>/w6; xmode /w6 type=80)&
```

A common complaint, and perhaps the reason why more people don't write packed Basic09 commands, is that Runb requires any passed parameters to enclosed in parenthesis and quotes. For example, to execute a packed command, you might have to type: cgp220sd ("/term","/p"). A royal pain. This shell recognizes packed procedures, and will do this automatically for you. Using the same example, you can just type: cgp220sd /term /p .

Similar to OS9/68000. Great for appending to logs or help files, or merging modules. Or for using the same temporary filename by overwriting.

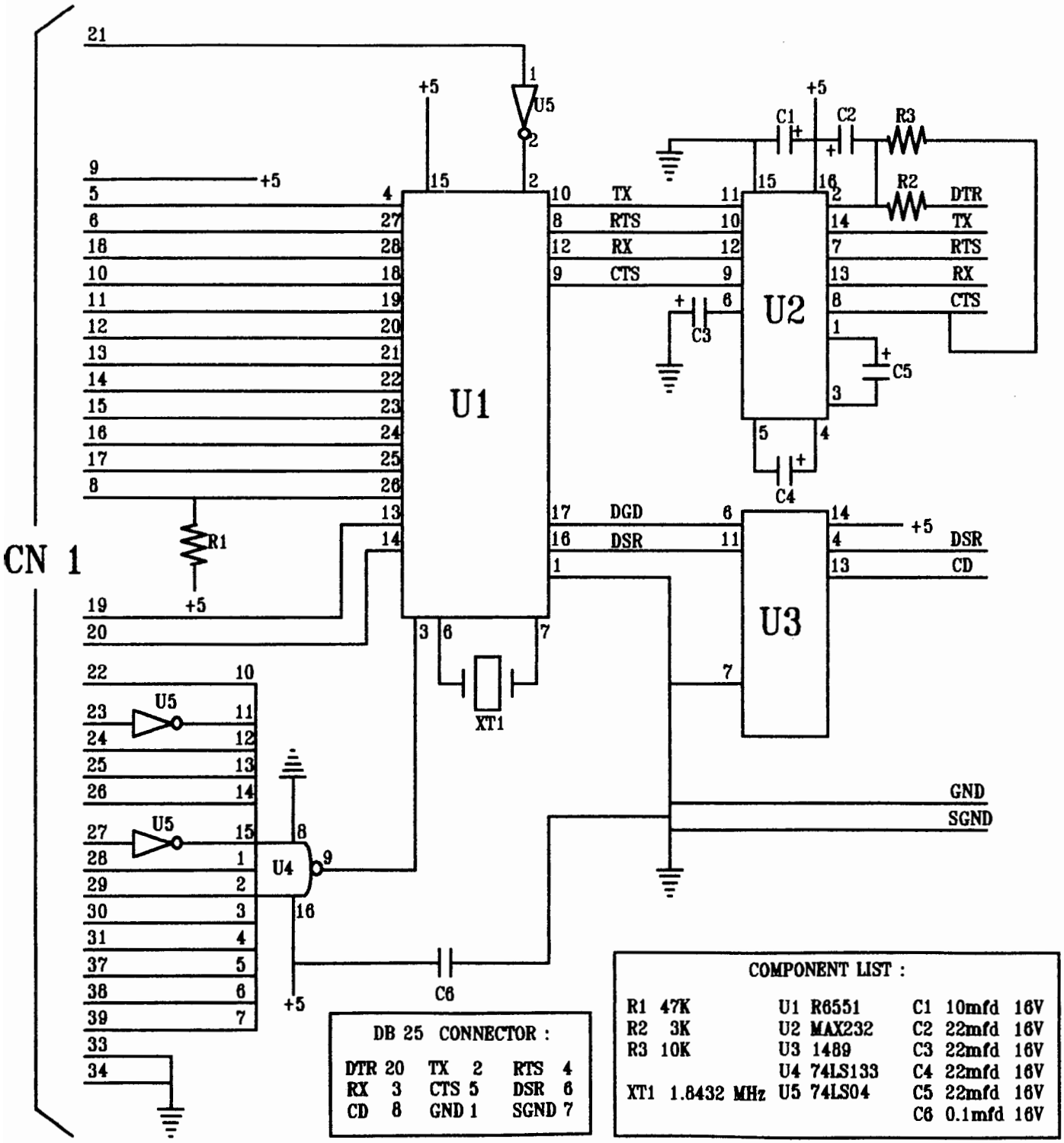
```
>+filename   - also >>+ and >>>+. Appends output to end of [filename].
>-filename   - also >>- and >>>-. Overwrites contents of [filename].
```

CD and CX may be substituted for CHD and CHX. PWD and PXD are now built into the shell as shell commands .PWD and .PXD.

Other features include memory scripts, command line logging, shell variables and processing, similar but more powerful than MSDOS variables, wildcard expansion, command priority modification, and much, much more.

If this material whets your appetite, we have the new shell module plus an expanded version of this documemtation (about 12 pages), available for distribution from the National User Group files using the normal arrangements as far as costs are concerned.

RS-232 Interface



DB 25 CONNECTOR :

DTR	20	TX	2	RTS	4
RX	3	CTS	5	DSR	6
CD	8	GND	1	SGND	7

COMPONENT LIST :

R1	47K	U1	R6551	C1	10mfd 16V
R2	3K	U2	MAX232	C2	22mfd 16V
R3	10K	U3	1489	C3	22mfd 16V
		U4	74LS133	C4	22mfd 16V
XT1	1.8432 MHz	U5	74LS04	C5	22mfd 16V
				C6	0.1mfd 16V

RS 232 Interface Board.
Introduced by Bob Devries.

WARNING ! This project is only for experienced hardware hackers!

On the previous page you will find a circuit diagram for an RS 232 interface board suitable for installing into either a Coco 1,2 or 3, or with a suitable card edge connector, it could be plugged into a MultiPak interface.
The circuit uses a 6551 ACIA chip as its heart. It requires only +5 volts and ground for a power supply, as the necessary positive and negative supplies are generated by the MAX-232 chip.
The ACIA chip is addressed at $FF68 and can be used with the ACIAPAK driver from OS9 (/T2 or /T3). There seems to be some problems with some of the public domain terminal programmes which run under RSDOS, but this problem can be overcome with a switch to disable the interrupt line from the interface.
Those of you who would like to build this board, and would like the added ease of using a circuit board, can purchase a PCB from :-

John Usher
47 Polaris Avenue,
Kingston. QLD 4114

The price is $20.00 plus postage. A component overlay will be included which shows all the connexions and straps and component orientations.
I have successfully used this board both in my Coco 2 (internally) and in my Coco 3 (in the multipak) and have run programmes like XCOM9, DeskMate under OS9, and Ultimaterm and GETerm under RSDOS. If running GETerm with the interface installed internally, the interrupt line from pin 26 of the 6551 chip must be disconnected.
Any technical queries can be directed to John Usher or me. Remember, however, that the National OS9 User Group takes no responsibility for the circuit, nor are we in a position to find faults and get things working if you make an error. As I said, it worked OK for me, I hope it does for you, too.

Regards,
Bob Devries.

oooooooooo0000000000oooooooooo
A DataBase in C
By Bob Devries

Here is the first installment of my programme 'DATABASE'. It was originally written as an assignment for my college course on C programming. The programme is a very simple name and address type database, allows no sorting to be done, but does allow most other database features such as edit, delete, insert and find. The database is disk based, and no records (except the currently displayed one) are kept in memory. If a record is deleted, it is set to 'null strings' and its position becomes available for future insertions. The first part of the source code is a header file 'ansi.h'. While this file could have been included in the main source file, making it separate made it suitable for porting to other machines. By changing this file I was able to compile the programme on MicroSoft Quick C with only one modification in the main file.

The functions in this section were the ones that were completely different for the MS-DOS machine and thus made a separate file necessary. The values for doing the various screen manipulations were obtained from the OS9 manuals. (They're there, take my word for it!) The programme was written for the level II 80 column screen, and I have not tried it on any other system. However, with small changes it would probably work OK on level I systems with either Opak or a Word-Pak 80 column card. I will examine those possibilities and perhaps mention patches to the code in a later article. Now here's the source for.....

```
/* ansi.h */
/* header file with routines for database programme */
cls()           /* clear screen function */
{
      printf("%c",12);
}
```

```
cursor(col,row)   /* position cursor at col and row */
int col,row;
{
        printf("%c%c%c",2,col+32,row+32);
}


revon()           /* turn on reverse printing (Level II only !!) */
{
        printf("%c%c",31,32);
}


revoff()          /* turn off reverse printing (Level II only !!) */
{
        printf("%c%c",31,33);
}


getch()           /* special unbuffered keyboard read routine */
{
        char ch;
        setbuf(stdin,NULL);   /* set stdin to unbuffered */
        setbuf(stdout,NULL);  /* set stdout to unbuffered */
        ch = getchar();       /* get the character */
        return(ch);
}



getarrow()        /* get arrow keys from keyboard */
{
        char ch;

        ch = getch();

        return(ch);
}


eraselin()        /* erase the line the cursor is on */
{
        printf("%c",4);
}
```

Please Note :- This file will not compile by itself, you'll need to wait for at least the next part before you can do that. Just type it in and save it in your C source directory for now. Also, not all the functions printed here are actually used in the database programme, but the header file was written for use as a general purpose header file.

RdV