For superior OS-9 performance, the

SYSTEM V

Provides a 68020 running at 25 MHz, up to 128 MBytes of 0 wait-state memory, SCSI and IDE interfaces, 4 serial and 2 parallel ports, 5 16-bit and 2 8-bit ISA slots and much more. The SYSTEM V builds on the design concepts proven in the SYSTEM IV providing maximum flexibility and inexpensive expandability.

AN OS-9 FIRST - the MICROPROCESSOR is mounted on a daughter board which plugs onto the motherboard. This will permit low cost upgrades in the future when even greater performance is required.

G-WINDOWS benchmark performance index for the SYSTEM V using a standard PC VGA board is 0.15 seconds faster than a 68030 running at 30 MHz with ACRTC video board (85.90 seconds vs 86.05 seconds).

Or, for less demanding requirements, the

SYSTEM IV

The perfect, low cost, high-quality and high performance OS-9 computer serving customers world-wide. Designed for and accepted by industry. Ideal low-cost work-station, development platform or just plain fun machine. Powerful, flexible and expandable inexpensively. Uses a 68000 microprocessor running at 16 MHz.

G-WINDOWS

NOW AVAILABLE FOR OS-9000

More vendors are offering G-WINDOWS with their hardware and more users are demanding it.

A PROVEN WINNER!

Available for the SYSTEM IV and SYSTEM V computers, the PT68K4 board from Peripheral Technology, the CD68X20 board from Computer Design Services and now, for computers running OS-9000 using 386/486 microprocessors.

OS-9/68000 SOFTWARE

CONTROLCALC - Real-Time Spread Sheet
SCULPTOR - Development and Run-Time Systems
DATADEX - Free Form Data Management Program
VED ENCHANCED - Text Editor
VPRINT - Print Formatter
QUICK ED - Screen Editor and Text Formatter
M6809 - OS-9 6809 Emulator/Interpreter
FLEXELINT - C Source Code Checker
IMP - Intelligent MAKE Program
DISASM_09 - OS9 Disassembler
PROFILE - Program Profiler
WINDOWS - C Source Code Windowing LibraryPAN UTILITIES

Distributor of Microware Systems Corporation Software

delmar co

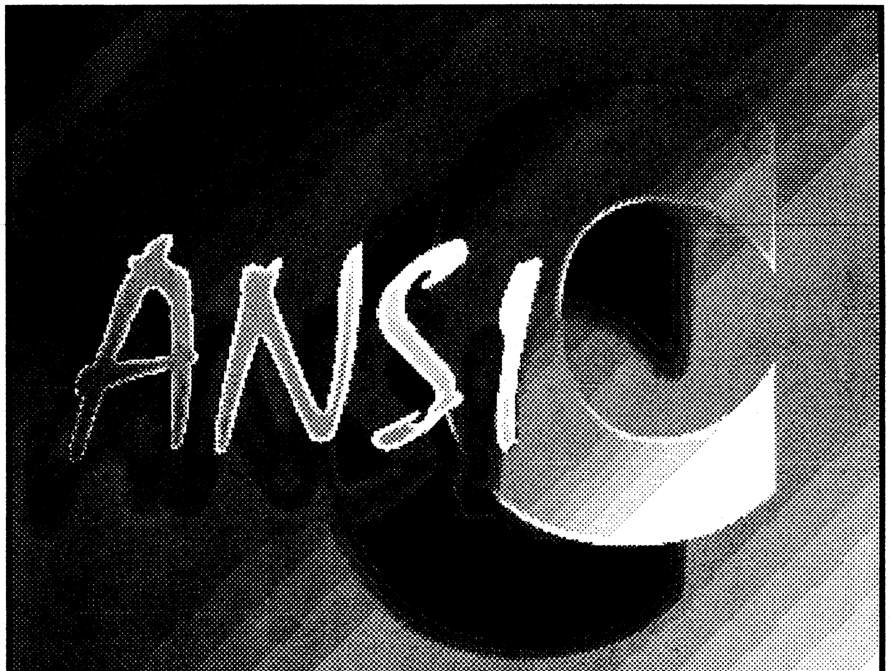
P.O. Box 78 - 5238 Summit Bridge Road • Middletown, DE 19709
302-378-2555 FAX 302-378-2556

Volume 1, Issue 9 • September 1993 $3.00

The "International" OS9 Underground®

A Fat Cat® Publication

Magazine Dedicated to OS-9/OSK Users Everywhere!



- Using Standard C
- What is ANSI C?
- Making C Programs More Efficient
- Computer Science 201
- Writing a Device Driver
- "OS9: The Q&A" is Back

New Lower Prices! from ColorSystems

Variations of Solitaire

Includes FIVE Variations, Pyramid, Klondike, Spider, Poker and Canfield. Complete documentation shows how to create your own games boot disk using special menu program which is included.

CoCo3 Version $29.95
MM/1 Version $39.95

WPShel

A Word Processing Oriented Point and Click Shell for all your word processing needs. Requires WindInt from your Multi-Vue Disk. Does not include editor, Formatter or Spelling Checker.

CoCo3 Only! $20.00

We accept Personal Checks or Money Orders drawn from US Banks or International Postal Orders.
NC residents please add 6% Sales Tax.
Call or write for a FREE catalog!

Please add $3 per item for shipping outside of the Continental United States.

NEW!

Using AWK With OS-9

A description of the AWK Programming language with an emphasis on GNU AWK for OSK. Includes the latest version of GNU AWK.

OSK Only! Just $19.95

OS-9 Game Pack

Includes FIVE complete games, Othello, Yahtzee, Minefield, KnightsBridge and Battleship. Includes special menu program and step by step instructions on creating your own games boot disk.

CoCo3 Version $29.95
MM/1 Version $39.95

All CoCo3 Programs require at least 256K of memory

Coming SOON! Indexed Files for OS-9 Level 2, OS-9/68000 and OS-9000!

ColorSystems
P.O. Box 540
Castle Hayne, NC 28429
(919) 675-1706

Quality OS-9 Software for the Color Computer 3 and MM/1 from IMS

Announcing

4th Annual *Atlanta* CoCofest

Holiday Inn, Northlake

October 2 & 3, 1993

Show Hours:
Sat Oct 2 10:00 AM - 5:00 PM
Sun Oct 3 10:00 AM - 3:00 PM

Vendor setup:
Fri Oct 1 6:00 PM - 11:00 PM
Sat Oct 2 8:30 AM - 9:45 AM

Admission : $5.00 (Whole Show)

Reservations:
Holiday Inn, Northlake, GA
(800)-465-4329 or (404)-938-1026

Sponsored by:
Atlanta Computer Society
PO Box 80694
Atlanta, GA 30366
BBS: (404)-636-2991

OS9 Underground Magazine Member Card

Participating Vendors

These vendors will offer the following discounts for OS9 Underground Member Card Holders

· CoNect	10% Off any order
· Sub-Etha	10% Off any order
· Canaware	10% Off WristSavers 10% Off WristSaver MousePad & 15% Off ENC9
· AniMajik Productions	20% Off all Software
· Fama Systems	10% Off any order

Software/Hardware Vendors...you can be listed here FREE! Contact the OS9 Underground for details. You need not be an advertiser for this Free service.

Be sure to give your card number when you place an order with these fine vendors. Not responsible for typos or mis-prints

Advertiser's Index

Vendor	Page
ColorSystems	IFC
Computer Design Services	7
Sub-Etha Software	7
CoNect	9, 26
OS9 Underground Subscriptions	14
Bob van der Poel Software	14
OS9 Underground Submissions	19
SBUG	20
Peripheral Technologies	29
DISTO	31
Atlanta Computer Society	IBC
DELMAR	BC

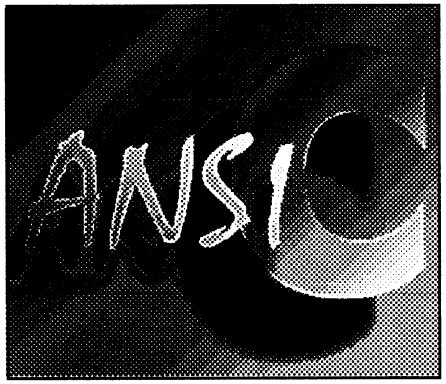
Please mention you saw it in the Underground, when you call these fine vendors...

The "International" OS9 Underground® Magazine

Dedicated to OS-9/OSK Users Everywhere

Volume 1, Issue 9

CONTENTS



The Underground Staff

Editor/Publisher:
Alan Sheltra
Assistant Editors:
Jim Vestal
Steve Secord
Technical Editor:
Leonard Cassady
Contributing Editors:
Vaughn Cato
Wayne Campbell
Allen Huffman
Eric Levinson
Scott McGee
Boise Pitre
Bob van der Poel
Typesetting/Layout:
AniMajik Productions

Fat Cat Publications and The "International" OS9 Underground Magazine and it's logotypes are registered trademarks. Subscription rates are $18.00 for 12 issues ($23.00 Canada, $27.00 overseas US Funds). Single or back-issues are available at the cover price (please call or write for availability). Fat Cat Publications is located at 4650 Cahuenga Blvd., Ste #7, Toluca Lake, CA 91602 · (818) 761-4135 (Voice), (818) 365-0477 (Fax) or (818) 769-1938 (Modem).

Mail Room Letters to the Editor

Shocked In North Hills

Dear Editor,

After having read the commentary that you put in the OS9 Underground, I am shocked that one person could have so much power that he could 1) call for new cabinet members (which apparently some/many responded to), 2) decide NONE of these people were qualified enough to run the OS9 UG, and then 3) disband the Users Group entirely.

This is too much power for one individual, and I'd heartily recommend that the charter be examined to see if Jim DeStefano had any rights whatsoever to perform all of these functions.

On a personal level, it may interest your readership to know that I, within the past 2 weeks, received from Jim DeStefano, a check in the amount of $10.73, for what was termed in the accompanying memo, my refund for my subscription to the 68XXX Magazine, which I was a subscriber to and Jim was the Editor/Publisher.

I find this curious (of course, I cashed the check!!), as Jim left me a message on Delphi about 6 months ago saying that the 68XXX Magazine was bankrupt and out of money, and that no refunds would be coming.

Considering the receipt of this check, and the return of monies to OS9UG people, too, I wonder if this check to me was indeed from Jim DeStefano, or was it from OS9UG funds.

I could be wrong in my assumptions and allegations aforementioned, and, if I am, publically apologize to Jim DeStefano.

If I am correct, however, then I think a precise audit should be made by someone in the OS9 community to make sure that 68XXX subscription refunds were not diverted from OS9UG funds.

-Jim Sutemeier, North Hills, CA
Former SysOp of <Plain Rap> BBS, which served CoCo/OS9 from 1984 to 1991.

In Need of a Dictionary

Dear Editor,

We have just received the latest issue of your magazine [July '93, Issue #7], and I am compelled to complain about the spelling and grammatical errors that appeared in our advertisement and that of Computer Design Services. As the creator of both these ads, I can assure your subscribers that neither was submitted in this sorry state; and, in addition, both ads had been altered substantially in format. As advertisements are a reflection of the companies that submit them, these can only give a very unfavorable impression.

This magazine is in dire need of professional proofreading. It is obvious that using a dictionary is a dying art, and the *front cover* even contained a spelling error. Yes, there definitely is a "y" in "everywhere".

Unfortunately, there has been an extraordinarily high morality rate in publications directed to the OS9/OSK/68XXX community. For such a publication

```
ENDIF
ENDWHILE
(* If the last line was less than 14 bytes,
(* it wasn't printed, so look for a comma at
(* the end, and remove it if its there, and
(* write the last line
IF MOD(count,14)>0 THEN
 IF RIGHT$(line,1)="," THEN
  line:=LEFT$(line,LEN(line)-1)
 ENDIF
 PRINT #outpath,line
ENDIF
(* Now, add the source code that will
(* read the data statements and write
(* the binary file to the current directory.
PRINT
PRINT "Adding ConvD code";
line:=""
cntr:=0
REPEAT
 READ line
 WRITE #outpath,line
 cntr:=cntr+1
UNTIL cntr=21
(* Now close the paths and end
CLOSE #inpath
CLOSE #outpath
CLOSE #hexpath
PRINT
(* Delete the temp file
DELETE "Hex"
END
(* The following data statements are the source
(* code necessary to convert the DATA
(* statements back to a binary file
DATA "CREATE #outpath,outfile:WRITE"
DATA " "
DATA "count:=1"
DATA "WHILE count<="+STR$(count)+" DO"
DATA "READ bite"
DATA "PUT #outpath,bite"
DATA "count:=count+1"
DATA "ENDWHILE"
DATA "PRINT"
DATA " "
DATA "(* Close Output Path"
DATA "CLOSE #outpath"
DATA "END"

(* Here's the error trap
er:=ERR
IF er<>215 AND er<>216 THEN
 CLOSE #inpath
 CLOSE #outpath
 CLOSE #hexpath
ENDIF
PRINT er
END
```

-Wayne Campbell

(EOF)

Moving?

If you are planning to change your address, you must let us know at least 30 days in advance in writing. Send a postcard or letter to **Fat Cat Publications** with your current address *and* your new address so we can update our records. We are not responsible for your subscription, if we are not notified in time.

Fat Cat Publications • 4650 Cahuenga Blvd. Ste #7, Toluca Lake, CA 91602

gotten a copy of the ConvD.B09 file), simply run Basic09 with a buffer large enough to load the file into and run. ConvD creates the file in the current directory, so be sure you are in the directory you want to use first. The same applies to ConvB.

There is a limit to how big of a binary file you can feasibly create with ConvB, simply because of the limitation of maximum buffer size of Basic09. (OSK users don't have this limitation). Level 2 users have a maximum buffer size of 40K available to them within Basic09. Any buffer greater than 32K will not allow the procedure to run in the buffer, and will require that ConvD be packed before running. If the ConvD file you have uses more than 39,000 bytes of the buffer, it may not pack. You'll have to experiment and see.

```
PROCEDURE ConvB
(* Convert Binary Files to B09 DATA Statements

PARAM infile:STRING[80]
DIM inpath,outpath,hexpath,bite:BYTE
DIM count,cntr,er:INTEGER
DIM char:STRING[1]
DIM hex:STRING[2]
DIM modname:STRING[29]
DIM line:STRING[80]

(* If any errors - close all paths, report error and
end
ON ERROR GOTO 10

(* Check parameter for pathlist
cntr:=LEN(infile)
REPEAT
 char:=MID$(infile,cntr,1)
 cntr:=cntr-1
UNTIL cntr=0 OR char="/"

(* If just a filename
IF cntr=0 THEN
 modname:=infile
 (* Otherwise get filename from pathlist
ELSE
 modname:=MID$(infile,cntr+2,LEN(infile))
ENDIF
(* Basic09 doesn't allow for straight conversion from
(* number to hex equivalent, so we must use the
(* PRINT USING statement, output to a temporary
(* file, to create the hex value of the byte read
CREATE #hexpath,"Hex":UPDATE
(* Open path to file to be read
OPEN #inpath,infile:READ

(* Create the file to be written to
(* ConvD means Convert Data to Binary
PRINT "Creating ConvD.B09 file for "; modname
CREATE #outpath,"ConvD.B09":WRITE

(* The procedure name must be the first line
(* in the file, or Basic09 will report a #43
(* error trying to read it
cntr:=0
REPEAT
 READ line
 WRITE #outpath,line
 cntr:=cntr+1
UNTIL cntr=3

(* Now we begin the task of data conversion
PRINT "Reading "; infile;
line:="DATA "
count:=0
WHILE NOT(EOF(#inpath)) DO
 GET #inpath,bite
 count:=count+1

 (* PRINT USING returns a string, so we
 (* write this string to the temp file,
 (* and then read it back in for inclusion
 (* in the data statement
 SEEK #hexpath,0
 PRINT #hexpath USING "h2",bite
 SEEK #hexpath,0
 READ #hexpath,hex
 line:=line+"$"+hex

 (* If 14 bytes have been read, terminate this data
 (* statement and begin the next one. This is
 (* because after the 14th byte is added, the B09
 (* editor would wrap the line if it was longer.

 IF MOD(count,14)=0 THEN
  PRINT #outpath,line
  line:="DATA "
 ELSE
  line:=line+","
-------------------------

DATA "PROCEDURE ConvD"
DATA "(* Convert Data Statements to Binary Data
File"
DATA " "
DATA " "
DATA "DIM outpath,bite:BYTE"
DATA "DIM count:INTEGER"
DATA "DIM outfile:STRING[29]"
DATA " "
DATA "(* Set Filename and Create File"
DATA "outfile:=""""+modname+""""""
DATA "PRINT """Creating """; outfile;"
```

to succeed, higher standards must become the norm. Your advertisers and subscribers have the right to expect a commitment to quality publishing from you.

-Carol Pap, Marietta, GA
Peripheral Technology

Editor-

As Editor/Publisher/Chief/Cook/Bottle washer of this magazine, plus holding 2 full-time jobs (one as Art Director, the other as free-lance graphic designer), it takes a lot of time, sweat, tears and love to put out each issue of the Underground. I certainly wish that someday I can afford to have a "full-time" proof-reader, but I'm afraid it's not quite in the budget as of now.

I do, however try my best to put together a good magazine for OS9, both in content and layout. Sometimes, I do overlook things; I am human.

I am constantly looking for ways to improve this magazine, both with better tools that let me do the work faster (thereby giving me more time for things like proof-reading and spell checking) and make it's appearance look better.

I do apologize for the typo that appeared in your ad and in compensation, have extended your ad insertion. I hope these human errors will not dissuade you from continuing to advertise in the OS9 Underground.

Still In Top Class...

Dear Editor,

I have just received and read from cover to cover issue #8 of The "International" OS9 Underground and thought I would send you my feedback {to} your magazine.

I thought that overall the issue was very good, not unlike the other ones, I found it to be in top class. I liked the two interviews that you published. The "One on One with Bob van der Poel" (I was sorry to hear that NASA couln't dish out the $60.00 for Ved) was very enthusiastic. I didn't know that Bob was getting so much attention from all over the world for his OS-9/680x0 stuff. That article told me that there is a market for those young OS-9 users out there who want to make a go at it. If Bob van der Poel can get such recognition just imagine what everyone else can get. By the way, that wasn't a put down.

I also liked "The Man Behind the Rocket" which I found very informative. But I wish that both interviews were a lot longer and covered more in regards to their views on OS-9, and why they use it and such.

One thing I have always wondered is how many of those developers and vendors make it a full time affair with OS-9.

Any, I liked the pictures that you published, well kind of, they were hard to distinguish, but none the less, a very nice touch.

By the way, in case you didn't know, I'm one of those very enthusiastic people when it comes to OS-9/680X0. Even though I don't own an OS-9/680X0 system, but a CoCo3 w/ OS9 L2, I do get those computer orgasms when I read or hear something very encouraging.

So here is my wish list:

Longer articles. The two page articles are just like Al Bundy's 30 seconds. They are excellent, but leave you wanting more.

Maybe a different theme for each issue. Not nescessarily OS-9ish, but maybe something more advanced than The Rainbow (RIP). Something like Byte [magazine] meets OS-9.

Maybe include some "Did you know...?" quotes/taglines. For example, "Did you know that NASA uses OS-9?" Stuff like that.

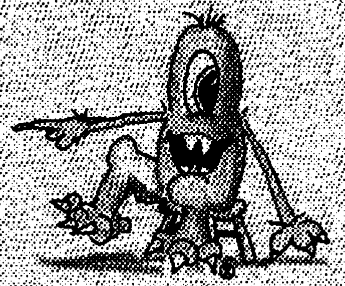
Maybe publish transcripts of computer fest seminars. Some of them seem to be very interesting and very good. [...]

Good luck. I liked the cover of issue #8, "OSK or Bust", very humorous.

-Don Vaillancourt,
Mississauga, Ont, Canada

Under is All It All (Editor's Column)

by Alan Sheltra (ZOG)



Reborn out of Chaos...

The OS-9 Users Group is, at the moment, on life-support. Rescued from total oblivion, by Carl Boll, Boisy Pitre and several other dedicated volunteers.

A conference was held in Chicago Saturday, August 14 to discuss the revitalization of the User Group. Many new ideas were discussed and road map sketched out for the reformation of the new group. I was not able to be there in person, but thanks to a teleconference system set up (thanks go to Scott Griepentrog) I was able to attend by phone.

Another meeting is scheduled to be held in Atlanta (coinciding with the Atlanta CoCo Fest) at the Northlake Holiday Inn. If you can make this conference, please do so.

A new national address has been set up for the OS-9 Users Group:

OS-9 Users Group, Inc.
6158 W. 63rd St.
Suite 109
Chicago, IL 60638

If you were a member in the month of July of this year, then your membership has been extended to the end of this year (refund received or not). You need not send any money, but donations are welcome.

If you were not a member during the month of July (again, this year), then you may renew your membership for $25.00 (US and Canadian), $30 (Foreign).

Help these guys out... these guys are helping YOU in the long run.

The Q&A Is Back...

In our search for a technical editor, we overlooked a talent right in our own midst. Our own columnist (C Software Engineering) Leonard Cassady has graciously filled that role. Leonard is qualified to answer both OS9 and OSK questions, as well as tackle your hardware questions as well.

Please send in those questions... there are many others who may benefit by what you need to know.

Questions or Feedback may be sent to Leonard, c/o the regular magazine address or email to: "ZOGster@delphi.com"

-Alan Sheltra (ZOG)

ConvB

by Wayne Campbell

ConvB is a utility designed to allow binary data to be converted to a form which can be included in a text listing, and can be reconstituted back into its original form at a later time.

An example use of ConvB is the inclusion of binary data in a program listing in a magazine (such as the OS9-Underground) which can then be converted back to its original form by the user.

ConvB is written in Basic09, and was developed on a Tandy Color Computer 3 running OS9 Level 2.

HOW IT WORKS

Usage:

```
Std.Shell: convb ("<pathlist>")
Shell+   : convb <pathlist>
```

where <pathlist> is a text string of up to 80 characters that may contain any valid filename or pathlist (including filename) to the directory where the file is stored.

ConvB reads the file specified byte-by-byte. It creates a file named ConvD.B09 that will contain the Basic09 procedure (including the DATA statements generated by ConvB) that will reproduce the file when run. (ConvB translates: Convert Binary to Data, and ConvD translates: Convert Data to Binary)

ConvD.B09 is ALWAYS the name of the file generated. Only the name of the file to be created is different within each different ConvD.B09 file created. Therefore, if you wish to save one, rename it or copy it somewhere where it will be protected before using ConvB again. ConvD is a Basic09 PROCEDURE that MUST be run in order to re-create the binary file from which it was derived!

To make ConvB as generic as possible, thereby ensuring it will run on most any OS9 system that has Basic, it uses no external subroutines at all. It determines end-of-file via a while/endwhile loop.

Each byte read is converted to a hex-value string, and then appended to a string that is destined to be a DATA statement in ConvD. Due to the way that Basic09 handles source line length, each DATA statement generated includes 14 bytes of the file being read. This way, there is no wrap around when listing the ConvD file from within Basic09.

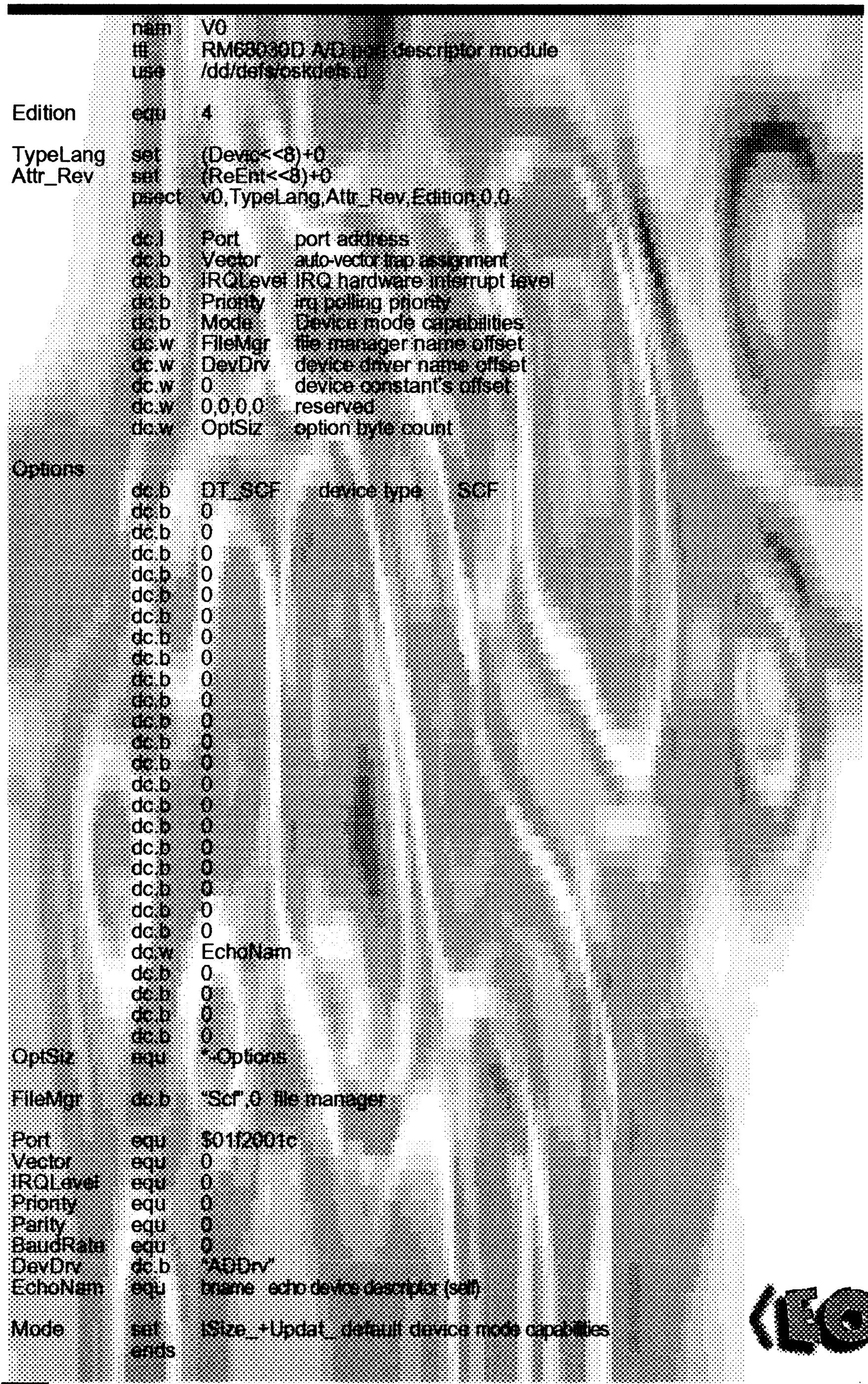
For those of you just beginning to learn Basic09, each B09 procedure must start with the line PROCEDURE <proc-name>, or Basic09 will report a procedure not found error (#43). So, this is first written to ConvD before anything else. Next, the DATA statements are generated and written to ConvD. When the last DATA statement has been written, ConvB writes the actual code that will cause ConvD to recreate the file from which the data was generated.

Note that ConvB uses the FILENAME as the name of the file to create, and NOT the MODULE NAME (for those who will use it for executables). This in no way affects the outcome of the file generated. However, the attributes of the file created are set only to —r-wr, so you'll have to re-attribute the file before attempting to execute it.

When passing a pathlist, the filename is derived from the pathlist so that the entire pathlist is not used for the filename to create.

The reason for putting the actual code last is that, without doing a system or function call, there's no way to know the size of the file until the last byte has been read. ConvD must know how many bytes of data it is to read from the data statements (ie. the size of the file) in order to avoid an error. So, the DATA statements are generated first, and the code added last.

To use ConvD to recreate the data file, (once you've typed the listing in, or have



```
               nam    V0
               ttl    RM68030D A/D port descriptor module
               use    /dd/defs/oskdefs.d

Edition        equ    4

TypeLang       set    (Devic<<8)+0
Attr_Rev       set    (ReEnt<<8)+0
               psect  v0,TypeLang,Attr_Rev,Edition,0,0

               dc.l   Port       port address
               dc.b   Vector     auto-vector trap assignment
               dc.b   IRQLevel   IRQ hardware interrupt level
               dc.b   Priority   irq polling priority
               dc.b   Mode       Device mode capabilities
               dc.w   FileMgr    file manager name offset
               dc.w   DevDrv     device driver name offset
               dc.w   0          device constant's offset
               dc.w   0,0,0,0    reserved
               dc.w   OptSiz     option byte count
Options
               dc.b   DT_SCF     device type    SCF
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
               dc.w   EchoNam
               dc.b   0
               dc.b   0
               dc.b   0
               dc.b   0
OptSiz         equ    *-Options

FileMgr        dc.b   "Scf",0  file manager

Port           equ    $01f2001c
Vector         equ    0
IRQLevel       equ    0
Priority       equ    0
Parity         equ    0
BaudRate       equ    0
DevDrv         dc.b   "ADDrv"
EchoNam        equ    bname  echo device descriptor (self)

Mode           set    ISize_+Updat_ default device mode capabilities
               ends
```

NEW 68020 COMPUTER BOARD!!!

The CD68X20 *sizzles* at 25MHZ - processing the most complex calculations in a *flash*!!!

CD68X20-25, 0K RAM	$699.00
Professional OS9/020 V2.4	$499.00
ULTRA C Compiler for OS9	$299.00
SCULPTOR V1.14:6 for Business Software Development	$ 79.00

Systems Available!

Computer Design Services
2550 Sandy Plains Rd. Ste. 320-234
Marietta, GA 30066
404/973-2170

Type'writer: n. archaic machine for writing by the impression of type letters; one of the reasons home computers were created.

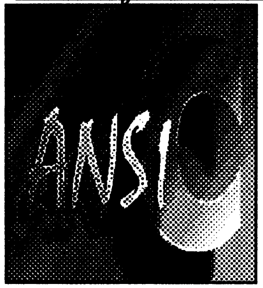
Write-right: n. a "real" word processor for the MM/1 by Joel Mathew Hegberg a complete word processor with "what you see is what you get" display of bold, *italics*, underline, and colors as well as a complete mouse-driven, cut-n-paste, on-screen text ruler, definable margins and tabs, insert/overstrikemodes, text formatting, word-wrap and definable printer codes.

much more: n. what else write-right has to offer for your 100% K-Windows compatible OSK computer.

Trash the typewriter. Purchase Write-Right today for $59.95 (plus $2.50 shipping and handling) direct from Sub-Etha Software. For more informationdownload the demo version ("WR_DEMO.LZH" found on GEnie or Delphi) or call (815) 748-6638.

Send to:

SUB-ETHA SOFTWARE
P.O. Box 152442 - Lufkin, Texas 75915



Using Standard C

by Vaughn Cato

C is very popular programming language. A few years ago, a standard was finalized by the American National Standards Institute (ANSI) that added many features to the original specifications. The ANSI standard was later accepted by the International Standards Organization (ISO).

This article describes some of the more important extensions to the language and how programmers can take advantage of them.

The original C specifications were given in a book written by Brian W. Kernighan and Dennis M. Ritchie called "The C Programming Language." This book has come to be called "K&R" and the definition of the language as given in this book is often called "K&R-C" to distinguish it from "ANSI-C". It is now also common to refer to the original specifications as "classic C" and the current standard as "standard C," and this naming convention will be used in this article.

SIMPLE EXTENSIONS

Many small but important changes were made to classic C to make it more practical and easy to use. In classic C, all names were only guaranteed to be unique within the first eight characters. This meant that you could have two variable names that were more than eight characters long, say "variable1" and "variable2," but they might actually be the same variable, since all the compiler had to be concerned with was the first eight characters. This works fine for small programs that only have a few functions. You just have to keep your names short, but with larger programs, code soon becomes very cryptic if only short names are used. In standard C, the number of unique characters in an identifier (an identifier is any name in a program) is extended to 32.

Another important change to C was to make all field names distinct within their own struct or union. In classic C, you aren't guaranteed that you can have two fields in two different structs or unions named the same thing. And many early compilers don't allow it unless the fields are at the same offset in the structure, so, for example,

```
struct a [
  int fa;
  int fb;
];

struct b [
  int fc;
  int fb;
];
```

is probably ok, since fb is the second int in both structures, but if you had:

Continued page 10

Simply copy the file to v1.a, change all references of v0 to v1 in the newly copied file, and change the address from $1F2001C to $1F2001D.

descriptor name	: t2
file manager name	: Scf
device driver name	: sc68681
port address	: $01f08001
irq vector	: 28
irq level	: 4
irq priority	: 5
device mode capabilities	: $23
device class	: $00
upper case lock	: 0
backspace option	: 1
delete line option	: 0
echo flag	: 1
automatic line feed flag	: 1
end of line null count	: 0
end of page pause flag	: 1
page length	: 24
backspace input character	: ^H
delete line character	: ^X
end of record character	: $0d
end of file character	: $1b
reprint line character	: ^D
duplicate line character	: ^A
pause character	: ^W
keyboard interrupt character	: ^C
keyboard quit character	: ^E
backspace output	: ^H
line overflow character	: ^G
parity code	: $80
adjustable baud rate	: $0e
output device name	: t2
xon character	: ^Q
xoff character	: ^S
tab character	: ^I
tab column width	: 4
driver global image offset	: 4

Figure 1 - Fields in an SCF device descriptor

Using r68, you can assemble the descriptors into a ROF:

```
r68 v0.a -o=v0.r
r68 v1.a -o=v1.r
```

Then, use l68 to link the ROF with the sys.l library to resolve any external references and produce the finished descriptor:

```
l68 v0.r -l=/dd/lib/sys.l -O=v0
l68 v1.r -l=/dd/lib/sys.l -O=v1
```

The descriptors are placed in the current directory and are ready for use.

Next month we'll study the evolution of the driver by writing the source code for several subroutines. Meanwhile, if you want more detailed information regarding drivers and driver writing, contact your local Microware representative about purchasing a copy of Dr. Peter Dibble's book, "OS-9 Insights," 2nd edition.

-Boisy Pitre

VENDORS NEED TO ADVERTISE... AND MAGAZINES NEED TO PAY THEIR BILLS...☺ SO, IF YOU'RE A VENDOR, LET'S HELP EACH OTHER OUT? AD RATES HERE ARE EXTREMELY REASONABLE, (YOU EVEN GET A FREE AD IF YOU'RE A FIRST TIME ADVERTISER) SO, LET'S TALK... CALL (818) 761-4135 OR WRITE FOR AN AD RATE CARD.

Writing a Device Driver Part II

by Boisy Pitre

We continue from last month's article in which we studied the various concepts and OS-9 system data structures related to driver writing. This month we'll examine our hardware device, decide how we will use it, and write the appropriate descriptors.

Choosing the Hardware

Depending on the hardware you wish to target, driver writing can range from simple to fairly complex. To keep our project simple, I've chosen a dual A/D converter as a hardware model.

In order to plan what our driver will do, let's look at some of the characteristics of the hardware:

- Dual A/D ports addressed at $1F2001C and $1F2001D
- This hardware doesn't support interrupts
- A port is initialized (ready for use) by writing byte $0
- Reading a byte value (0-255) from either port returns a byte value corresponding to the voltage (0 to 5v)
- Writing a byte value (0-255) into either port produces a corresponding voltage (0 to 5v)
- A port is terminated by writing byte $FF

The reader should be aware of the simplicity of this particular driver. Most devices are more complicated in design and require particular attention to detail. Our target device allows us to write a basic read/write driver with setstat/getstat routines without becoming bogged down in a complex state diagram.

There are many applications for an A/D converter - for this series we will target our driver to support (1) basic reading and writing to/from the device, and (2) sound sampling to/from the device.

Creating the Descriptors

For our device, we'll need two device descriptors: one for the left channel and one for the right channel. Let's call these descriptors 'v0' and 'v1' respectively. Writing a byte to v0 will cause a distinct voltage to appear on the left channel. Likewise, writing a byte to v1 will do the same on the right channel. Reading will work much the same way, except that the A/D chip will provide us with a number between 0 and 255 which corresponds to the voltage on the respective input channel.

Figure 1 shows the fields in an SCF device descriptor (in this case, t2). Notice that in our descriptor source code (figure 2), most of the fields hold a value of $00. The reason for this is inherent in how SCF interprets data going to and coming from the driver.

When using the I$ReadLn or I$WritLn system services, SCF's line editing routines may modify the data. By setting the option fields in the descriptor to $00, no interpretation is done to the data. Ideally, we should only communicate with the device through I$Read and I$Write calls since these bypass SCF's line editing routines and obtain data in raw (as is) form.

Although the source code listed here is for the v0 descriptor, we can easily adapt it for the other channel of our A/D driver.

CoNect *New Hardware*

Mini-RS232 Port: If you are in the market for a Tandy-compatable serial port, check out our Mini! This ROMPak unit supports all seven control lines available, and can supply more output current than even the Tandy Pak! Jumper selectable addrees and cd swap.

Only $49.95 (Y Cable use requires 12 volt power supply, add $9.95)

XPander: The XPander allows you to assemble the most compact CoCo3 system possible using a stock motherboard. For example, the electronics of my 2 meg CoCo3 with Tandy Floppy Controller, Burke & BurkeXT, WD1002 Hard Drive controller, rs232 port, puppo and Hi-Res adaptors froms a block 12 inches long, 7 inches deep and 3 inches at it highest point.. Not only will this fit in the smaller PC cases,, but in a modified CoCo case.

Obviously this is not a full tilt MultiPak clone - there just isn't room. The two external slots may both contain /scs decoded devices, but only one slot ROM may be used. The external slot may be used either as a ROMPak port (disables internal hardware when Pak is inserted), or as an undecoded buss slot. 12v is available at all slots.

The no-slot RS232 port is a virtual clone of the mini-rs232 described above, and saves not only a slot but quite a bit of room in the finished package.

The Xpander is available in two versions. If a PC type case/power supply will be used, order just the board. CoCo Kit includes a new lower case shell and 450ma +/-12v power supply.

CoNect Custom CoCo Cables!

Cassette - (mini or rca)	$ 4.95	Comp. Video (6 feet)	$ 3.95
Disk Power (either way)	$ 5.95	Printer (DIN to DIN)	$ 4.95
2 Drive Floppy Data	$14.95	3 Drive Floppy Data	$19.95
RS232 (db 25, db9, din4 to db25,db9 - normal, nulmodem, or dcd swap)			$ 9.95

Ram Upgrades

64K CoCo 1 (F board), or CoCo2 (2 or 8 chip) with instructions	$ 7.95
512K CoCo3 (various makes) not always available	$49.95
2 Meg CoCo3 (Disto) with SIMMS $194.95 Installed (add 11.95 for 6309)	$219.95
Hitachification Install CPU socket and 6309, 6809 returned	$29.95

CoNect
449 South 90th Street
Milwaukee, WI 53214
(414) 258-2989 • Delphi: RICKULAND • Internet: rickuland@delphi.com

```
struct a {
int fa;
int fb;
};

struct b {
int fb;
int fc;
};
```

you could have problems. It turned out to be a real problem keeping all field names distinct when you have a large program with many structs and unions. In standard C, you are guaranteed that each struct or union has its own set of field names that doesn't interfere with any other names in the program.

One restriction placed on structs and unions in classic C is that they can't be copied around like other types. This means you need to use memcpy to copy a struct or union, and you can only pass a pointer to a struct or union between functions instead of being able to pass the whole thing around. C has always been considered a low-level language, and copying large objects around could be very inefficient. In general though, it isn't reasonable to restrict these operations as they are sometimes useful, so in standard C, they are all allowed.

VOID

In C, functions are always functions, which means they are always expected to return something. Sometimes, though, you want to have a function that doesn't return anything. To do this in classic C, it is common to leave the return type off the function and use a plain "return" or no "return" at all.

```
f()
{
  ...
  return;
}

- or -

f()
{
  ...
}
```

Leaving the return type off of a function doesn't mean it doesn't return anything. Like many things in C, if you leave the type off it means int.

If you called this function and did something with the return type you could have problems.

```
{
  int a = f(); /* What is the value of a? */
}
```

In standard C (and many more recent non-standard compilers), a special return type can be used called "void"

```
void f()
{
  ...
}
```

Now the function is said to return "void," which basically means it doesn't return anything. Now, if you try to use the return value of "f", you will get an error.

Another use of "void" is the void pointer. When programming in C, you sometimes need a pointer that doesn't point to any particular type. This can happen if you want a certain variable to hold a pointer to many different types. In classic C, it is common to use a char pointer for this purpose. With standard C, compilers are expected to be a lot better about checking

if you can find this one, (its been out of print for many years, but still one of the best discussions I've seen), "Floating Point Computation" by Pat Sterbenz, Prentice-Hall Inc. 1973.

Q: *On page 2-2 of the C compiler manual (coco version), it states that information (such as version number or copyright notice) can be placed between the module name and the executable code of the object module by use of the "info" directive, and that the #asm switch can be used in c programs to include this directive.*

When I tried to use this, the .a output showed that the info lines were directly after the nam statement, which (I assume) is the correct position. However, when I assembled the file, c.asm reported a bad mnemonic error on the lines that had the directive "info" in them.

What am I doing wrong?

- Wayne Campbell, Sepulveda, CA

A: Without seeing the actual code, (including the ".a" code generated), it would be difficult to give an accurate answer.

The "NAM" mnemonic is a string literal constant and should have no "whitespaces" or should be enclosed in parentheses to delimit the string boundaries. Since the "#asm" is a C preprocessor directive and the info lines appear in the ".a" file, you are using the directive properly. The problem appears to be the format of the information you're including in the ".a" file isn't in the proper format recognized by the assembler compiler, "c.asm".

One possibility is that CoCo C manual assumes you are using the Level 2 Relocatable Macro Assembler, (rma), that comes with the Development System. If I recall correctly, the "c.asm" module is a carry over from Level 1 and the two assemblers are different.

In order to use the "rma" with the C compiler package, you simply rename the "rma" to "c.asm", otherwise you have hack the "cc" module to use "rma" instead of "c.asm".

Please contact me through email and we'll see if we can isolate the problem.

If you have a question, please send a letter or postcard to:

The OS9 Underground
OS9: The Q&A
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

or send your question in email to either of the following addresses:

"zog!leonard@abode.ttank.com"
"ZOGster@delphi.com (to Leonard in the subj line)

(EOF)

DISTO

2-Meg Kit (0K)	$ 99.95
SCII	$100.00
3in1	$ 75.00
HDII	$ 75.00
Full Turn of the Screw	$ 20.00
Schematics	$ 20.00

Send certified check or money order to:

DISTO
1710 - Depatie, St. Laurent
Quebec, Canada H4L 4A8
S/H - $4.00 for one item and $6.50 for two or more items.

Technical Editor:
Leonard Cassady

An Introduction...

First, I'd like to thank everyone for their concern and well wishes concerning my recent surgery. Everything went as planned and there were no complications. I'll be back to my full health shortly.

For those readers who don't know me, a short bit of background is in order.

My first hands-on experience with computers came when I purchased a "grey case" 4k CoCo. Once home, the first thing I did was void the warranty to see what was in there. The next morning found me down at the local electronics store buying 16k memory chips.

This lit a fire under me that is to this day, still burning. I enrolled at the local college, and after a few semesters, landed a job working with a imaging systems firm as a Quality Assurance Software Test Engineer, (long job title, isn't it?), where I stayed for several years.

I program in Basic, Pascal, some Assembler, but my first love is the C Language. Having been around electronics most of my life, (my father was an engineer himself and worked mostly on NASA projects), I've never been shy to hack hardware.

I'll endeavor to answer all questions, software or hardware related, addressed to this column. Some research may be required for the more complex inquiries, so if your question does not appear right away, it means I'm trying to insure the answer is as accurate as possible.

-Leonard.

Q: *What is the difference between MSBIN & IEEE floating point numbers and what kind of floating point numbers does the CoCo 'C' compiler use?*
- Dean Lieber, Tarzana, CA.

A: Both Motorola and, (assuming 'MSBIN' means MicroSoft Binary), Intel have implemented one, or part of one, of the IEEE Standards for dealing with FPP, (Floating Point Precision).

They differ in "byte order" storage, or which byte they consider to be the "first" one in a larger piece. Byte ordering depends on the microprocessor chip architectures. Motorola uses "left-to-right", or "big endian" storage and Intel uses "right-to-left", or "little endian" storage.

```
Big Endian,   {     MSB      }{     LSB      }
left to right  |00|01|02|03||04|05|06|07|
bit fields

Little Endian, {     LSB      }{     MSB      }
right to left  |07|06|05|04||03|02|01|00|
bit fields
```

The difference should be of no importance, since the hardware platform and associated compilers adjust 'in coming traffic' to the implementation's requirements.

If you plan on doing more with bit field manipulation, such as encoding for a telecommunications protocol, or data file encryption, there are several references available that dive into much greater detail.

I'd suggest "The C programmer's guide to Serial Communications", which has somewhat complete description of bit field encryption.

The ANSI/IEEE Standard 754-1985 contains the most common implementation of Floating Point Precision used today, and

types of things, so you would always have to cast your pointer to a char pointer.

```
void CopyEveryOtherByte(char *d,char *s,int num,bytes)
{
  ...
}

void f()
{
 int data1[10];
 int data2[10];
 CopyEveryOtherByte((char *)data2,(char
*)data1,10*sizeof(int));
}
```

This could be annoying, and using a char pointer is not really representative of what you are trying to do. The void pointer solves these problems. You can assign a pointer to any object (but not a function) to a void pointer, and assign a void pointer to any other object pointer. This gets around the problem of having to cast, and also allows you to explicitly state your intent that it isn't one particular type of pointer you want. Indirecting a void pointer is illegal, since it doesn't point at anything, it just points.

PROTOTYPES

Along with these fairly simple extensions, some new features were added to the language to help keep the programmer from making mistakes. One of these, and the one that standard C is best known for, is prototypes. In classic C, functions are always declared like the following:

```
int f(a,b)
int a;
float *b;
{
  ...
}
```

This means that the function has two parameters, the first being an int, and the second being a float pointer. When you call the function, you are expected to pass it two parameters of the appropriate type, but it isn't an error to pass it something else. You can write

```
f("hi there",5,2);
```

and you won't know there is a problem until you run your program and it fails in some way. It is possible for the compiler to check this for you and give you some kind of warning, but classic C compilers rarely do this, and if the function is in another file, then there is no way to check it at all since the declaration of this function would just be:

```
int f();
```

This doesn't give any information on the number of parameters or their types.

Under standard C, a new way of declaring functions is available , and it looks like this:

```
int f(int a,float *b)
{
}
```

With this style of function declaration, you put the types of the parameters along with the names in the parenthesis. This is called "prototype form."

A prototype form declaration is called a "prototype." When you want to declare a function that exists in another file, you can now specify the number and type of these parameters as well:

```
int f(int,float *);
```

Now, the compiler can (and must) check to make sure you pass the correct number and type of parameters to the function. By putting the declaration of a function in a header file and including this header file in all filesthat refer to the function, you can be sure that the compiler will tell you if you pass the wrong thing to a function. You need to include the header file in the file where the function is defined as well, so that the compiler will check to see that your declaration and the definition match.

If the function doesn't have any parameters at all, however, there is a problem. If you were to declare one of these functions it would look like:

```
int g();
```

The compiler would think this was just an old-style function declaration and not do any checking at all. For this, a special syntax is used.

```
int g(void);
```

This explicitly tells the compiler that there are no parameters to the function. You need to define the function in a similar way:

```
int g(void)
{
   ...
}
```

There is one other little issue with prototypes. There are some functions that don't have a specific number of parameters. The best example is the "printf" function. The first parameter is always a char pointer, but, based on what is in the string, it can have any number of parameters after that. For this, a special syntax is used:

```
int printf(char *,...);
```

This way, the compiler will make sure you pass a char pointer as the first parameter, but will let you pass any number of parameters of any type, or no parameters, after that. It is up to you to be careful in these cases and pass what is expected.

CONST

With standard C, the concept of unchanging objects is introduced. This is specified with the keyword "const." It can be used in various ways:

```
/* a has a value of 5 that should never change */
const int a = 5;

/* q points to p, and should always point to p */
const int *const q = &p;

/* p points to an int that shouldn't change */
const int *p;

/* f promises not to change the string passed to it */
void f(const char *p);
```

Note, that these uses of const are meant to express an idea. "a" shouldn't be changed, "p" shouldn't change what it is pointing to, etc. What they are doing is informing the compiler what the intentions of the programmer are. If someone else modifies the code it will be apparent that they shouldn't change "a," and if a function returns a const pointer, it will be apparent that that function doesn't want you to change the data the pointer points to. This can be very important in large programs where it becomes difficult to remember what you had in mind at the time, or if you are working on a project with other programmers, it can really help relay intentions. The syntax for "q" is a little odd. This is the syntax used to indicate that the pointer "q" is itself unchanging. So that

```
int a;
int *const pp = &a;
```

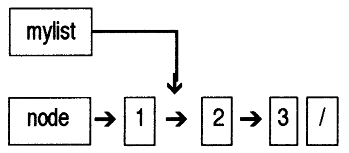
indicates that pp points to an int that can change, but that pp itself shouldn't.
Const pointers used in a function signature have an interesting meaning. They basically say that they "promise" not to change the

This sets node to the first item in the list (if there is one) and then, the while loop will execute as long as node points to a list node. As soon as we reach the end of the list, node is assigned a null and the while loop ends. Inside the loop, we do whatever is desired for that node (such as "printf("%d\n", node->data);" to print it's data) and then point node the the next element.

If we are finished with a list, and want to free it's memory, we use a very similar loop to delete the whole list.

```
node = mylist;
while(node != NULL){
        mylist = node->next;
        free(node);
        node = mylist;
}
```

In this code, we first set node to the first node in mylist as before. Again, the while loop continues until node is NULL and will have node point to each element in the list. The first line of code in the loop will point mylist to the next item in the list, leaving the one pointed to by node out of the list.



The second line frees this node, and the third line updates node again. You have to be careful here NOT to use "node = node->next;" as the last line like you did in the traversal loop. This is because node has been freed, and is not a valid pointer any more.

Once the loop executes once, the first node, with data set to "1" is gone and both node and mylist point to the node who's data is set to "2" which is now the first node in the list (or the "head" of the list). Each time the loop executes, the head is freed and the next node becomes the head.

Well, today's lecture time is about up, so we'll have to leave inserting and deleting a single node until next time. Your homework assignment for today is to write a program that will ask the user for data (a simple int will do, but you might try a string instead) and put each item entered into a linked list until some end indicator (a zero or an empty string) is entered, then will print each item in the list out and free it's memory before ending. Feel free to make use of my email address above if you have problems. If you don't have an email account, contact "The OS-9 Underground" for a list of STGNet BBS's that you can use to contact me.

Class dismissed!

-Scott McGee

68XXX COMPUTER PRODUCTS from Peripheral Technology

- a company with a reputation for quality!

PT68K4-16, 1MB	$299.00
PT68K2-10, 1MB	$199.00
ALT86 for PC Compatibility	$199.00
Profess. OS9	$299.00

1480 Terrell Mill Rd. #870
Marietta, GA 30067
404/973-2156

First, the declaration:

```
typedef struct LLIST *llist;
struct LLIST{
    int  data;
    llist next;
}
llist mylist;
llist node;
llist *node_address;
int  counter;
```

Now, the code to fill it.

```
mylist = NULL;
node_address = &mylist;

for(counter = 1;counter<=3;counter++){
    node = (struct LLIST *) malloc(sizeof(struct
        LLIST));
    node->data = counter;
    node->next = NULL;
    *node_address = node;
    node_address = &node->next;
}
```

After executing this code, the list would look like this:

$$\begin{tikzcd} \boxed{\text{mylist}} \arrow[r] & \boxed{1} \arrow[r] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \boxed{/} \end{tikzcd}$$

where mylist points to the node containing 1, and that node's next points to the node containing 2, who's next points to the node containing 3, who's next is NULL (the '/' means null).

OK, lets talk about the code for a bit. To start with, we have a for loop that runs the variable counter from 1 to 3. In each iteration of the loop, we do five things. First, we use malloc() to get memory for the new node. We include a type cast to make the pointer returned by malloc() match the type of node. Second, we set the data element of the new node to the value of counter. This is just so we can tell the nodes apart by their data. Third, we set the next element of the new node to null. This is so we don't ever accidently think it points to something valid.

Fourth, we have to put the new node into the linked list. Since node_address is set to the address of mylist (&mylist) before the loop starts, *node_address is the same as mylist. Therefore, the line "*node_address = node;" will assign our first node to mylist. Finally, since we want to assign the next new node to the next pointer of this node, we must point node_address at it. Therefore, the line "node_address = &node->next;" is executed.

The really tricky part of this is the use of node_address to always point at the location where you want to put the next node. Doing it this way lets us take the numbers out of the "for" control statement and put in a variable while knowing that our code will keep track of it. Without this, we would have to assign the second element to mylist->next and the third to mylist->next->next and so on. This would work for just three nodes as we have here but would get very difficult to code for more and would not work at all if you don't know how many node you'll have until you run it.

Another way to do this that isn't quite so tricky is to replace the last three line inside the loop with the following two:

```
node->next = mylist;
mylist = node;
```

which will put each new node at the front of the list and the rest of the list will follow it. This, however, will reverse the order of the list as it is created. Both are correct but sometimes (as we will see in future classes) the use of something like node_address in necessary.

To traverse (or walk down) the list, the following code will do the job.

```
node = mylist;
while(node != NULL){
/* insert code here for whatever you want to do to each node */
    node = node->next;
}
```

data pointed to. This way, the programmer can look at the declaration of the function and see that it is safe to pass a pointer to an object they don't want changed to the function. The declaration for "strcpy," for example can use this to make it clear that the string being copied won't be affected.

```
char *strcpy(char *,const char *);
```

With all uses of const, if you try go against const promises you have made, you will get an error, but since C never wants to get in the way when you really know what you are doing, you can cast pointers to const objects to pointers to non-const objects, so

```
const int a = 2;

void f(void)
{
  int *p = (int *)&a;
  *p = 5;
}
```

is legal. As long as you don't use casts, the type system will pretty much guarantee that an object declared const won't change. The address of a const object is a const pointer. Indirecting a const pointer gives you a const object. You can't assign to a const object, and you can't assign a const pointer to a non-const pointer:

```
void f(void)
{
  const int a = 5;
  int *p = &a; /* error - &a is a const pointer */
  const int *q = &a; /* ok */
  *q = 7; /* error - *q is const */
}
```

ENUMERATIONS

One other extension to classic C is enumerations. An enumeration allows you to define a set of names that have their own type. An enumeration looks like this:

```
enum colors { black, red, green, yellow, blue,
        magenta, cyan, white };
```

This means that all the names given are of type "enum colors." Anywhere you might normally put "int" you put "enum colors" instead to indicate that this variable holds the value of one of the listed names.

```
enum colors favorite = blue;
```

The values of these names are integers. The first name has a value of 0, the second, 1, the third 2, and so on. Enumerations can be used any time you want to name a set of values to be used in identifying something. The values are constants, and can be used anywhere a normal integer number can be used:

```
char used_color[white+1];
```

A special syntax lets you set the value of an enumerated name to any integer you want. This allows enumerations to be used in some places instead of #defines:

```
enum {
MAX_USERS=100,MAX_MESSAGES=10000
};
```

When you make names have specific values this way, any names that are not specificly set after that will increase sequentially from that number, so

```
enum { first=1, second, fourth=4, fifth };
```

means that "first" has a value of 1, "second" has a value of 2, "fourth" has a value of 4, and "fifth" has a value of 5. Note that the name of the enumeration itself is optional.

CONCLUSION

Standard C offers many advantages over classic C, and any classic C program will still work on a standard C compiler, except in cases where the old compiler let you get away with things you shouldn't have done, and the new compiler doesn't allow it. The addition of prototypes and const add greatly to the safety of the language. Other extensions just make it easier to get the job done.

Standard C compilers are available on OSK machines, and a new program by yours truly called "ansifront" is available for OS9 in beta release that will translate standard C into classic C and provide all the error checking of a standard C compiler.

- Vaughn Cato

<EOF>

GET YOUR OWN COPY!

Are you reading someone else's Underground?

Shame on you!

Did you know 12 issues (1 per month) are Only $18/Yr (US) ($23 Canada, $27 overseas)
Mail your check or M.O. to:
The "International" OS9 Underground
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602
(818) 761-4135

Ved/68000 Text Editor 2.0

Our editor just got better! Ved 2.0 now includes an integrated spelling checker! Plus it supports multiple buffers! This is in addition to all the features of the original: user control over macros, key bindings, editing modes; automatic indenting and numbering; wordwrap on or off; search; find/replace; block move/copy/delete; word and line delete; "undo"; and a lot more!

Ved 2.0 also supports your KWindows mouse... but it still works with any terminal (as long as it has cursor positioning).
Ved comes complete with MVEF (an editor for creating the environment files Ved uses for configuration) and a 100 page manual which fits your Microware binders. For more information, just drop us a note and we'll send you full information on Ved and other fine products.

Ved 2.0 complete with the spelling checker and MVEF, costs only $59.95 plus $3.00 shipping and handling. To order please send your check or M.O. and preferred disk format to:

Bob van der Poel Software

P.O. Box 355	P.O. Box 57
Porthill, ID	Wynndel, BC
USA 83853	Canada V0B 2N0
Phone (604) 866-5772	CIS: 76510,2203

COMPUTER SCIENCE 201
INSTRUCTOR:
SCOTT McGEE

CS 201 - Introductory Computer Science

Course Requirements: A basic programming knowledge is needed. Knowledge of C is of extra benefit, but I will attempt to do most of this in pseudo-code that can be applied to any good programming language.

It has come to my attention that number of very good programmers in our community are not only entirely self taught, but actually lack knowledge of some of the basics presented in introductory Computer Science classes.

Since many of these fine programmers seem interested in learning these concepts, I will be presenting them in this series of articles. Since these concepts will be taken straight from my experience as a Teaching Assistant in the Computer Science classes at the University of Utah, I am entitling these articles "CS 201". While you will not be able to get college credit for these course, you will pick up information that may help your programming skills.

To contact me for consultation, (all instructors must provide office hours of some sort!) you may reach me as shown below:

email: Internet:
- smcgee@cymru.UUCP or
- smcgee%cymru@uunet.uu.net

UUCP:
- uunet.uu.net!cymru!smcgee

STGNet:
- smcgee@os9er

snail mail (US Mail):
Scott McGee
c/o The OS-9 Underground
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

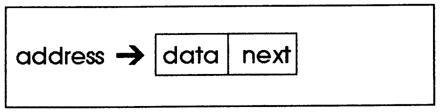
LESSON 1: LINKED LISTS

Any discussion of linked lists will usually involve the use of pointers, pointer dereferencing, and include the use of malloc() and free() to get and dispose of memory for the list. If you are not well versed in these, either consult another source to acquaint yourself, or let me know if you think a refresher course is needed.

A linked list is a method for storing data. It is used much like an array would be used. The major difference is that linked lists are dynamic in nature and can shrink or expand to meet your needs, while an array is created with a specific size that can not be changed.

An element of an array really has two things associated with it, the data (whatever is stored in the array) and the address (the array index). A linked list also has data and an address. The data is essentially the same as that of an array, whatever you declare it to be. The address is, however, different. The address of a linked list element is a pointer to the memory where it is stored. Since this memory need not be (and often is not) sequential, you cannot therefore just add one to the current address to get the next. For this reason, a third item is present in a linked list element, the next address.

To help make this a bit clearer, lets draw a picture:



What if there is no next element? Well, then next is set to NULL. To see this in more detail, lets actually declare a linked list and put the numbers 1 - 3 in it.

(I'll use 'C' here, but don't assume that this is a complete program!)

CoNect
Used and Surplus

Atari Joystick Adaptor	$5.95	Color Mouse	$12.95
CoCo Cheap Sticks (pair)	$3.95	Speach/Sound Pak	$24.95

Other hardware is available in limited quantities

Software:

Stock varies, but these titles were in that last cataletter!
AlphabetZoo, ChildPace, CoCoChecker, CoCoMax3, Color Profile, Dallas Quest, EDTASM+, Elite Word, EZBase, FastDup, FlightSim, GwanaBwana, Graphicom, Handsoff, Investograph, KidWriter, Max10, OneonOne, Pitfall2, Pitstop2, RadioBall, Rocky's Boots, Sand of Egypt, Shamus, Spectaculator, Telewriter 64, War Game Designer, Arkanoid, Castle Tharrograd, Chess, ColorFile2, Diagnostics, Downland, Dungeons of Daggorath, Football2, RadWarrior, Rampage, Shanghai, Springster, Tetris, Thexder, Bedlam, Colormath, Pooyan, Pyramid, RaakiTu, Basic09, C Compiler, Cave Walker, CoCoartist, DeskMate, DeskMate3, Developement System, DL Logo, Donald Duck's Playground, DynaCalc, Flight Simulator2, Home Publisher, Interbank Incident, King's Quest 3, Koronis Rift, Mickey's Space Adventure, Micro Illustrator, Microscopic Mission, Multivue, Level1, Level2, Pascal, Patch09, Phantomograph, Profile, Recue on Fractalus, Rogue, RSB, Screen Dump Utility, Sub Battle Simulator, TSEdit, TSSpell, TSWord, Winnie the Pooh, Where in the World is Carmen SanDiego, Wiz/WizPro, Zone Runner, Begginers Guide to Windows, Basic09 Tour Guide, Color Computer Assembly Language Programming, MousePad
We'd Like to buy you unneeded CoCo stuff to add to the list.

Prices on used and surplus goods depend mainly on availability. To find out exactly what's in stock, you need a subscription to the 'cataletter'. This part catalog/part newsletter is published bi-monthly, and is availble for the very low price of... nothing! Just get your address to CoNect by any means available.

It is impossible to predict the availability of these items. In general, a single piece won't even make the cataletter- much less a magazine ad. Drop us a line with your needs and we will tag the inventory file- you get first shot!

Please include $4.00 shipping and handling
All orders are shipped Priority Mail. New hardware is warranteed for 90 days

CoNect
449 South 90th Street
Milwaukee, WI 53214
(414) 258-2989 • Delphi: RICKULAND • Internet: rickuland@delphi.com

C Software Engineering

by Leonard Cassady

What is ANSI C?

The C language is a member of ALGOL family of algebraic programming languages. Similar to PASCAL, ADA, and PL/I, and with increasing popularity, it is now widely used for applications. Due to the uniformity of the implementation of the language subset, it is easily ported to many different hardware platforms.

The traditional language reference is "The C Programming Language" by Brian Kernighan and Dennis Ritchie, published in 1978. This reference has often been called "Traditional C", "Modern C", "Pre-ANSI C" or "K&R Standard". In 1982, the American National Standards Institute, (ANSI), formed a subcommittee and seven years later the ANSI X3.159-1989 standardization was adopted, thus, the "ANSI C Standard" was born.

The ANSI C Standard does not require a programmer to write portable code, however, some effort was made to make the ANSI C Standard and the POSIX Standard, (Portable Operating System Interface for Computer Environments IEEE Std. 1003.1-1988), compatible.

The ANSI C Standard is by no means a minor change to the C language, (it is almost twice as large as the K&R Standard). However, it is fortunate that most programs written in any version of C use well-understood features of the language, and are easily ported.

Simply stated, programs written to be 'strictly conforming' to ANSI C must NOT depend on unspecified aspects, undefined aspects, or hardware dependent implementations that are not defined in the Standard. Other than this restriction, (if it can be called one), there are a several additions and differences between the ANSI C Standard and the Traditional C versions which a programmer should be familiar.

PREPROCESSOR

Originally the C preprocessor had nine pre-defined directives. Two additional directives became common to the C environment before the ANSI C Standard, and the ANSI C Standard introduced four new directives in addition to perceived Traditional Standard for a total of fifteen directives.

(Fig 1)

#define	- Define a macro.
#undef	- Remove a macro definition.
#include	- Insert text from another source file.
#if	- Conditional constant expression test.
#ifdef	- Conditional macro "true" definition test.
#ifndef	- Conditional macro "false" definition test.
#else	- Alternate action for conditional macro test.
#elif	- Now common to C. Alternate action for secondary macro test.
defined	- Now common to C. Preprocessor function that yields an integer one, (1), if a name is defined as a preprocessor macro, or defined as zero, (0), otherwise. Used with the '#if' and '#elif' directives.

(Fig 1 Cont)

#endif	- Terminate conditional macro test.
#line	- Supply a line number for compiler messages.
unary #	- New in ANSI C. Token conversion to string constants.
binary ##	- New in ANSI C. Merging token string constants.
#pragma	- New in ANSI C. Specifies implemen -tation-dependent information to the compiler.
#error	- New in ANSI C. Produce compile-time error with a designated message.

PREDEFINED MACROS

The ANSI C Standard specifies a total of five pre-defined macro constants. Each of these built-in macro constants begin and end with an underscore. They can NOT be undefined or redefined and the preprocessor directive 'defined' cannot be used with them. Additionally, arguments to them are not allowed, except '__LINE__' and '__FILE__', (you may write to these two by using the preprocessor directive, '#line'). Other Implementations may define additional macros. The macros, '__LINE__' and '__FILE__' are now common to non-ANSI environments. (Fig 2)

TOKEN CONVERSION

(unary # operator)

During macro expansion in ANSI C, the unary '#' token and the macro name appearing within the macro definition are replaced by the corresponding argument(s) enclosed in string quotations. Each whitespace in the macro's expansion is replaced by a single space character and any backslash characters are preceded by a backslash character to preserve their meaning.

Example:

```
#define TEST(a,b) printf( #a ">" #b "=%d\n", (a) > (b) )
```

If we set 'a' to '21' and 'b' to '\n',the call:

```
        TEST( 21, '\n' );
```

Expands to:

```
        printf( "21 > '\\n' = %d\n", (21) > ('\n\') );
```

(Fig 2)

Macro	Value
__LINE__	Expands to a decimal integer constant that returns the current line number of the source program. You can alter the line number by writing a '#line' directive.
__FILE__	Expands to a string literal constant that returns the current filename of source file name. You can alter the filename by writing a '#line' directive.
__DATE__	Expands to a string literal constant that returns the date the preprocessor was invoked. The format is: 'Mmm dd yyyy'. The month name, 'Mmm', is the same as for dates generated by the library function, 'asctime()', declared in 'time.h'. The day part, 'dd', ranges from '1 - 31' and a leading zero becomes a 'space' character.
__TIME__	Expands to a string literal constant that returns the time the preprocessor was invoked. The format is: 'hh:mm:ss', which is the same for dates generated by the 'asctime()' library function declared in 'time.h'.
__STDC__	Expands to the decimal integer constant one, (1). The preprocessor should provide another value, or leave the macro undefined, when invoked from a non-ANSI C Standard environment.

```
IF a$="N" OR a$="n" THEN 15
ON ERROR GOTO 25
OPEN #printer,"/p":WRITE
IF deposit THEN
PRINT #printer,TAB(32); z1; "."; cnts
ELSE
PRINT #printer,TAB(2); bankacct(y); TAB(24);
ENDIF
SHELL "Date >/p"
PRINT #printer
IF deposit THEN
PRINT #printer \ PRINT #printer,TAB(32); z1; ".";
cnts
ELSE
PRINT #printer," "; bankname(y); TAB(41); "*";
z1; "."; cnts
PRINT #printer
PRINT #printer,CHR$(27); CHR$(20);
PRINT #printer,o$
PRINT #printer,CHR$(27); CHR$(19);
ENDIF
CLOSE #printer
IF deposit THEN
total:=total-FLOAT(z2)/100-z1
ELSE
total:=total+FLOAT(z2)/100+z1
ENDIF
GOTO 15
25 PRINT "Printer not ready. Please make
ready."
GOTO 20
PROCEDURE english
REM ENGLISH (C) 1984, E. Levinson, All rights
reserved.
PARAM z:INTEGER; a$:STRING[40]
IF z>999 THEN
PRINT "Error: Range is 0-999"
END
ENDIF
DIM number(19):STRING[10]
DIM tens(9):STRING[10]
DIM x,y:INTEGER
FOR x=1 TO 19
READ number(x)
NEXT x
FOR x=1 TO 9
READ tens(x)
NEXT x
REM Get arrays of strings
a$=""
x=z
15 IF x<20 THEN
REM if number is less than 20, use weird "teen"
and single digit numbers
a$=a$+number(x)
ENDIF
IF x>19 AND x<100 THEN
REM Use combination of tens and single digit
words.
a$=a$+tens(INT(x/10-.5))
IF MOD(x,10)<>0 THEN
a$=a$+"-"+number(MOD(x,10))
REM we must hyphenate combinations to be
politically correct
ENDIF
ENDIF
IF x>99 AND x<1000 THEN
a$=number(INT(x/100))+" HUNDRED"
REM if number is over 99, affix hundreds to the
string.
y=x-INT(x/100*100)
REM get decimal portion remaining
IF y<>0 THEN
a$=a$+" AND "
REM More political correctness (and better
grammar too!)
x=y
GOTO 15
ENDIF
ENDIF
END
DATA
"ONE","TWO","THREE","FOUR","FIVE","SIX","SEVEN","EIGHT","NINE"
,"TEN","ELEVEN","TWELVE","THIRTEEN"
DATA
"FOURTEEN","FIFTEEN","SIXTEEN","SEVENTEEN","EIGHTEEN","NINETEEN"
DATA
"TWENTY","THIRTY","FORTY","FIFTY","SIXTY","SEVENTY","EIGHTY"
,"NINETY"
DATA "HUNDRED"
REM Are these spelled right?
REM I am tired.
```

<EOF>

```
PROCEDURE checks
REM PRINTCHECKS (C) 1985 by E. Levinson
DIM month,day,year:STRING[2]
DIM dayt:STRING[8]
DIM a$:STRING[40]
DIM o$:STRING[80]
DIM z1,z2:INTEGER
DIM cnts:STRING[2]
DIM x,y:INTEGER
DIM amt:REAL
DIM dol,cnt:INTEGER
DIM bankname(40),bankacct(40):STRING[40]
DIM printer:INTEGER
DIM path:INTEGER
DIM total:REAL
DIM deposit:BOOLEAN
x:=1
total:=.0
cnts=""
PRINT "Reading /dd/sys/accounts"
ON ERROR GOTO 10
OPEN #path,"/dd/sys/accounts":READ
WHILE EOF(#path)=FALSE DO
READ #path,bankname(x)
READ #path,bankacct(x)
x:=x+1
ENDWHILE
CLOSE #path
ON ERROR
dayt:=LEFT$(DATE$,8)
year:=LEFT$(dayt,2)
month:=MID$(dayt,4,2)
day:=RIGHT$(dayt,2)
GOTO 15
10
IF x=41 THEN PRINT "Error: Maximum 40
accounts (80 lines) allowed." \ ENDIF
PRINT "Error: /dd/sys/accounts is non-existant or
format is improper."
PRINT "File must alternate payee and account
numbers.  For example:"
PRINT  \ PRINT "Visa"
PRINT "4121260512345678"
PRINT "Master Card"
PRINT "5121367212345678"
PRINT
PRINT "If an account number is not desired,
leave that row blank, but be sure it"
PRINT "exists."
END
15
deposit=FALSE
FOR y=1 TO x-1
PRINT "<"; y; "> "+bankname(y)
NEXT y
PRINT
PRINT "<"; y; "> Not listed here"
PRINT "<"; y+1; "> Deposit slip"
REPEAT
INPUT "Enter Bank # or 0 to exit? ",y
IF y=0 THEN
PRINT  \ PRINT USING "'Total of checks
$',r10.2<"; total
END
ENDIF
UNTIL y<x+2
IF y=x THEN
INPUT "Pay to: ",bankname(x) \ INPUT "Account
number: ",bankacct
(x)
ENDIF
IF y=x+1 THEN deposit=TRUE \ ENDIF
INPUT "Enter Dollars $",z1
PRINT "Enter Cents $"; z1; ".";
INPUT "",z2
IF z1=0 AND z2=0 THEN 15
IF z2<10 THEN cnts="0"+STR$(z2)
ELSE
cnts=STR$(z2)
ENDIF
IF NOT(deposit) THEN
IF z1=0 THEN
RUN english(z2,a$)
o$=a$+" CENT"
IF z2>1 THEN o$=o$+"S" \ ENDIF
ELSE
IF z2<>0 THEN
RUN english(z1,a$)
o$=a$
RUN english(z2,a$)
o$=o$+" DOLLAR"
IF z1>1 THEN o$=o$+"S" \ ENDIF
o$=o$+" AND "+a$+" CENT"
IF z2>1 THEN o$=o$+"S" \ ENDIF
ELSE
RUN english(z1,a$)
o$=a$
ENDIF
ENDIF
PRINT  \ PRINT bankname(y); " $"; z1; "."; cnts
PRINT o$
ENDIF
20
INPUT "Print? <CR>=Yes",a$
```

TOKEN MERGING

(binary # # operator)

New tokens are formed by the presence of a 'merging' operator, '# #', in macro definitions. The tokens surrounding any '# #' operator are combined into a single token.

Example:

```
#define NAME( extension )  test. # # extension
```

If we set 'extension' to 'bak', the call:

```
NAME( bak );
```

Expands to:

```
test.bak
```

CONDITIONAL COMPILATION

(The '#elif directive)

The '#elif' command is analogous to the 'else if' conditional construct in C. The expression following a '#elif' statement must be a constant expression and is converted to a 'long int' data type. Macro calls may be used in the constant expression.

The '#elif' directive is convenient way to make code a bit more readable by eliminating the need for nested '#if'...'#else'...'#endif' constructs, although it is functionally the same.

Example:

```
Traditional C                 ANSI C

#if OS9                       #if OS9
# define BUFFER 256           # define BUFFER 256
#else                         #elif OSK
# if OSK                      # define BUFFER 1024
#   define BUFFER 1024        #else
# else                        # define BUFFER 512
#   define BUFFER 512         #endif
# endif
#endif
```

THE 'defined' OPERATOR

The 'defined' operator is analogous to the '#ifdef' conditional construct in C, and may be used only in '#if' and '#elif' expressions.

Complex expressions may be built using the 'defined' operator.

Example:

```
#if (defined(OS9) && !defined(BIGBUF)) ||
    (defined(OSK) && !defined(BIGBUF))
#  define BIGBUF 32676
#endif
```

The traditional use of the '#ifdef' and '#ifndef' directives would require several lines of code to produce the same effect.

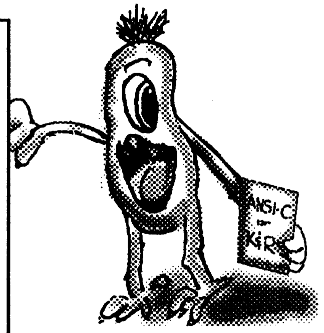
THE '#pragma' DIRECTIVE

The '#pragma' directive performs implementation specific tasks or may add new preprocessor functionality to the compiler. The argument to '#pragma' is subject to macro expansion. At this time, there are no standard pragmas, although several are coming into common usage.

Example:

```
#if defined(OS9) && !defined(__STDC__)
# pragma custom(abs)
#endif
```

Consult your compiler documentation for any pragma implementations.



THE '#error' DIRECTIVE

Any sequence of tokens may follow the '#error' directive, and after macro expansion, are sent to the standard error device, (usually the terminal). It is most useful for detecting inconsistencies, or illegal conditional compilation values, and producing diagnostic messages.

Example:
```
#if INTSIZE < 16
# error "Defined 'INTSIZE' too small!!!"
#endif
```

An attempt to compile a file with the compiler option:

```
cc -DINTSIZE=8 test.c
```

will produce the error message:

```
Defined 'INTSIZE' too small!!!
```

PREPROCESSOR LINE FORMATTING

Pre-ANSI C compilers usually required the preprocessor directive, '#', to be the first character in a new line, and no 'whitespaces' are allowed between the pound sign and the preprocessor command. The older preprocessors treat the preprocessor command, '#', followed by a whitespace as a 'null directive', and would either be treated the same as a blank line, or passed to the compiler as erroneous code.

The ANSI C Standard permits whitespace to precede and follow the preprocessor directive, '#', on the same source line. The only restriction is that the pound sign must be the FIRST non-whitespace, or non-tab, character on the line in order to recognized as a preprocessor directive.

Example:
```
#include <stdio.h>
/* Traditional and ANSI compatible */

#           include <stdio.h>
/* causes error in Traditional C,     */
/* allowed in ANSI C.  */

        #include <stdio.h>
/* causes error in Traditional C,  */
/* allowed in ANSI C.  */
```

Preprocessor lines are recognized before macro expansion, and if a macro expansion 'looks' something like a preprocessor directive, it will NOT be recognized by the preprocessor. Some UNIX implementations violate this restriction.

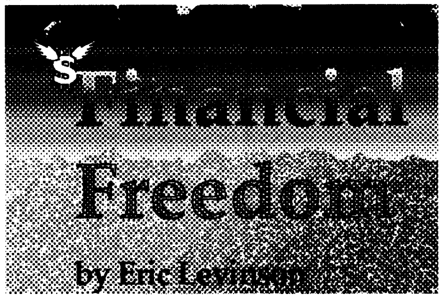
Example:
```
#define GETLOCAL #include <locale.h>

          GETLOCAL
```
Will expand to:
```
# include < locale . h >
```

The result is erroneous code passed to the compiler, due to the whitespaces in the header name.

Also, if a line ends with a backslash character, '\', the following line will never be recognized as a preprocessor command, and any preprocessor directives will be ignored.

Example:
```
          #define BACKSLASH \
          #define PERIOD        .
```

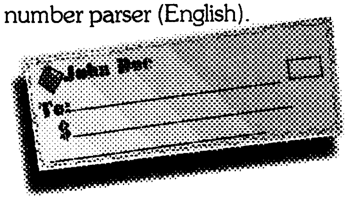
The second preprocessor is treated as a comment, or 'null directive'.



How many checks a month do you write? Before I paid my car off, canceled my cable T.V. subscription and paid off my department store credit cards, I found myself writing about 10 checks a month, to the same businesses scattered on different days of each month. This program will push the burden of writing checks onto your computer. The only thing you need to do with the check is sign it.

Many people don't know that several businesses will allow you to change your billing date. This will allow you to print all your checks in a batch. You have a pile of bills all due on or around the same day. It would take a simple call to each creditor, utility company, and your bank for the car loan. It then becomes a less dreaded task when your computer prints the date, account number, amount, payee and spells out the amount of the check for you. Depending on your checks (size, formatting and the like) you won't have to order special checks at a ridiculously high rate to fit your pin-feed printer.

I have been using this program for quite some time. It works great with standard billfold side or top tear off checks. I personally prefer the duplicates, that way a check register is not needed. It is quite simple in design, however the numeric to English converter took some thinking. This program is divided into two procedures, the main one (checks) and the English producing number parser (English).



The program is relatively easy to set up and use. The first thing you need to do is set up your accounts file. This is a file called /dd/sys/accounts. It is a plain text file with alternating payees and account numbers. You place on the first line the payee's name, on the next line your account number with the payee and so on to a maximum of 40 (80 total lines). Each line can contain up to 40 alphanumeric symbols. If you run the program without an accounts file it will give you an error message and an example of the correct format for this file. Run the program . You are presented with a menu of 1 through n accounts plus n +1 (Not listed) and n+2 (Deposit slip). If you have an occasional bill (like to pay for a glass window your son broke) you can use the Not listed option and manually put in a payee and account number. If you are using standard checking deposit slips, the deposit slip will print single entry deposits. I am using a Tandy DMP 106 printer for the output. The formatting is preset for this printer, but obviously could be changed to suit your taste.

Now when I do my checks, it takes about 15 minutes to print, sign and mail. I don't have to worry about missing a payment, or take about 15 minutes 10 times a month to write my checks. Enjoy!

-Eric Levinson

Eric Levinson is a Senior in Computer Science at Sonoma State University in Rohnert Park, California and works for Broderbund Software as a Technical Support Representative in Novato. He has been programming with OS9 on a Tandy Color Computer since 1983.

```
moveq.l #0,d0    pre-extend 32 bits
move.b 1(sp),d0
bsr foo
```

This time, a 32 bit load of 0 is used to clear the most significant 24 bits of D0 before the value of 't' is moved to the least significant 8 bits. About the same code size and timing as earlier. However, if we change the type of 't' to int we get the following:

```
move.l (sp),d0
bsr foo
```

There is a trade off here...it takes a tad longer to increment a long counter than a short or byte.

Here's another interesting example. First off, assume that 'a' and 'b' are global variables of type byte. The following function:

```
main()
{
   int t;

   ....

   if(a<t) foo(b);
}
```

compiles to something like:

```
move.b a(a6),d0    grab 'a'
ext.w d0           sign extend
ext.l d0
cmp.l (sp),d0      compare 'a' and 't'
bge _6
move.b b(a6),d0    grab 'b'
ext.w d0           sign extend
ext.l d0
bsr foo
_6
```

Again, too compare 'a' to 't' we first have to sign extend 'a'...and when foo() is called, 'b' has to be sign extended (it even gets worse if you compare 'a' to 't' and then pass 'a' since 'a' then gets sign extended twice).

Compare this to the code generated when 'a' and 'b' are of type int:

```
move.l a(a6),d0
cmp.l (sp),d0
bge _6
move.l b(a6),d0
bsr foo
_6
```

So...what are the lessons here?

• First, don't be too quick to assume things. Your program has to be translated into assembly language and, even with an optimizing compiler, there are tradeoffs to be made.

• Second, remember that chars are always "promoted" to ints in a function call. If a variable is going to be tossed around a lot, it's probably best off being an int.

• Third, when comparisons between different sizes of variables are done the compiler has to first convert them to the same size. This takes code and time. You are best off keeping all variables to be compared the same size.

By following these guidelines I was able to shave approximately 3000 bytes off a 98,000 byte binary. I assume that the result also runs a bit faster...but, with a text editor on a 680xx it is pretty hard to tell.

These types of optimization may or may not be compiler specific. If one of you has an Intel processor, or a 68xxx compiler for something other than OS-9, or if you want to fool around with the same questions on a 6809...let us know if the results differ. As always, I enjoy hearing from you. You can contact me at:

PO Box 355, Porthill, ID 83853 or
PO Box 57, Wynndel, BC, Canada V0B 2N0
or Compuserve 76510,2203.

- Bob van der Poel

However, the following is perfectly legal.

```
Example:

#define setflag(str1, str2) if((strcmp(str1, str2)) == 0) \
   flag = TRUE
```

In this case, the preprocessor treats the line break as a comment, and line breaks within comments do not terminate preprocessor directives.

RECURSIVE MACRO DEFINITIONS

Macros appearing in their own expansion are not re-expanded in ANSI C, either immediately or through a sequence of nested macros. Marco names are only recognized within the macro body ONLY after the macro body has been expanded. This allows the programmer to re-define a function in terms of the its old definition.

```
Example:
        #define plus(a,b) add(b,a)
        #define add(a,b) ((a) + (b))

        main()
        {
        int c = 0, a = 1, b = 2;

        c = plus(1, 2);
        }

 Expands to:
        c = plus((plus(1,2),0);
        or
        c = (add(0,plus(1,2)));
        or
        c = ((0) + (plus(1, 2)));
        or
        c = ((0) + (add(2, 1)));
 or finally
        c = ((0) + (((2) + (1))));
```

Traditional C versions usually provide no mechanism to prevent infinite macro recursion. Some versions do not even allow recursive macro calls.

SIDE EFFECTS IN MACRO ARGUMENTS

Macros can produce side effect problems. In the following examples, (one is a function, the other, a macro), the macro will produce an erroneous result due to re-expansion.

```
function:

        int square(a)
        int a;
        {
        return (a * a);
        }

the statement:

        a = 3;
        b = square(a++);

becomes:

        b = (3 * 3);
or
        b = 9;
```

SUBMIT!

Your submissions are always welcome. Programs, Tutorials, News, User Group and Fest Info. Call (818) 761-4135 for details or email: ZOGster@delphi.com Submissions may be sent to the email address above or to:

The OS9 Underground Submissions
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

The variable ‘a’ is post-incremented to the value 4. When the function, ‘square’, is used, the argument expression is evaluated only once, so any expression side effects occur only once.

However, if we use the same statement with the macro definition:

```
#define SQUARE(a) ((a) * (a))

main()
{
a = 3;
b = SQUARE(a++);
}
```

the macro expands to:

```
      b = ((a++) * (a++));
or
      b = ((3) * (a++));
```

Variable ‘a’ is then incremented after the first usage to the value 4 due to the post-incremental operator, and the macro then becomes:

```
b = ((3) * (4));
```

due to the post incremental side effect and evaluates as:

```
b = 12;
```

since the variable ‘a’ has been post-incremented twice. No combination of macro argument binding can correct this side effect.

First in a Series

This is the first installment of several in a series on the additions and differences introduced by the ANSI C Standard. One of the goals of this series will be to develop a working ANSI C Standard Library for the OS9/OSK environment using the ANSI C nomenclature.

Any comments, suggestions are welcomed and should be sent to:

Software Engineering Assoc.
Standard ANSI C Library
6530 Independence Ave Ste #168
Canoga Park, CA 91303

or emailed:
“zog!leonard@abode.ttank.com” or
“ZOGster@delphi.com” (to Leonard in subject line)

(EOF)

SBUG

Your source for upgrading your Color Computer hardware and extending your software library...

Call us for more information...

SBUG
1251 W. SEPULVEDA, STE 400
TORRANCE, CA 90502
(310)539-9702
AFTER 7:00 PM PST

LOOK FOR US AT THE ATLANTA COCOFEST IN OCTOBER!!!

Making C Programs More Efficient

by Bob van der Poel

I’ve spent a bit of time lately honing my Ved Text Editor (the 68K version) and have found some interesting ways to make a the program more efficient by using the “right” variables.

But first, a few ideas I had on what size of a variable to use:

1. If I was going to count a small loop, it would make sense to use a char (8 bit) variable as a counter. Not only would the char save some memory over an int, it would also be quicker since a char takes fewer clock cycles to increment or decrement.

2. If I have a number of small variables which fit into an unsigned char or short integer, using those (instead of an int) would save program and data space.

Well, it ain’t always so....

To make discoveries on this level for yourself you should write short program code and compile them to assembly language —the results may surprise you.

first test program is:

```
woof()
{
   char t;
        for(=0;t<10;t++) foo();
}

,1(sp)
 _7
 cmpi.b #10,1(sp) :6
 blt _5
 ...
```

the program a bit:

```
woof()
{
   char t;
        for(t=0;t<10;t++) foo(t);
}
```

Now, we are passing the variable ‘t’ to the function foo(). Have a look atwhat happens to the assembly language code (I’ve cut the output and addedcomments to make this a bit more clear):

```
...
 bra _7
_5
 move.b 1(sp),d0   grab the value
 ext.w d0          sign extend 8 to 16
 ext.l d0          sign extend 16 to 32
 bsr foo
_8
 addq.b #1,1(sp)   increment counter
_7
 cmpi.b #10,1(sp)  loop till we've done 10 times
 blt _5
...
```

Note the ext.w and ext.l commands. These sign extend the 8 bit value into a 32 bit integer (which is what we pass on to the function). Changing the type of ‘t’ from char to unsigned char makes an interesting change to the code: