GREAT OS-9 PROGRAMS

UD-CACHE II ***Light Speed Disk Cache*** ***$149.00***

Improves disk access performance 2 to 100 times.
Usable with any RBF device – supports multiple devices, multiple sector access, variable sector size.
Installs in a minute – no system generation required.

XSCF ***SCF Extender File Manager*** ***$60.00***

Improves I$ReadLn line editing – cursor backup/advance, overwrite/insert, associative recall.
Usable with any SCF path an any terminal – G-Windows support coming soon.
Installs in a minute – no system generation required.

LSrcDbg ***Source Level Debugger Launcher*** ***$50.00***

Routes debugger's and application's output to different terminal screens.
Requires only one keyboard common to both the debugger and application.
Greatly ilmproves debugging efficiency of screen-oriented applications.

Disk Squeezer ***Fragmented Disk Reorganizer*** ***$295.00***

Reorganizes fragmented disk contents and files for better access performance.
Eradicates Error #000:217 – Segment List Full.
Comes with utilities to check disk and file fragmentation.

IBF ***IEEE488 File Manager*** ***Ask***

Supports IEEE488 interface bus with popular LSIs at file manager level.
HP compatible library functions.
IBF needs proper porting to particular hardware platforms. Inquire for list of existing portings or licensing conditions for new porting.

NEW BOOK

The OS-9 Guru ***Vol 1: The Facts*** ***$59.00***

A new book authored by Paul Dayan of Galactic Industrial,England.
Covers basics to detailed OS internals – a "must" reference for both beginners and hackers.
420+ pages, paperback. $59.00 includes postage. Delivery may take 3-4 weeks.

*All programs work on any OS-9/680x0 system (V2.2 & up). Exceptions: LSrcDbg requires a secondary terminal or window and SrcDbg from Microware; IBF requires IEEE488 interface hardware and properly ported device driver. *UD-CACHE II's disk access improvement factor varies depending on the physical device and applications.

S&H: US (48 states) orders add $4.00 for ground service or $11.50 for FedEx 2nd day air; AK, HI and outside US ask for quotation. CA residents add 8.25%. Send check or money order (no charge cards or CODs) with preferred disk format (important). Reach us by fax or mail.

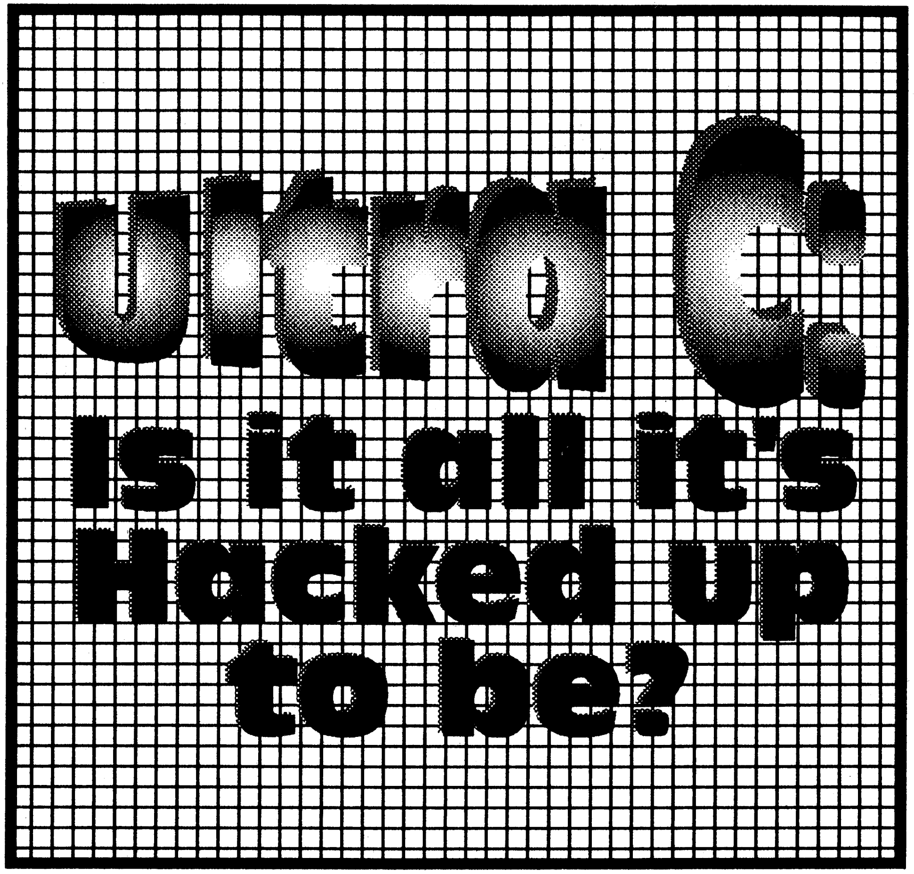
ARK SYSTEMS

ARK Systems USA
P.O. Box 23
Santa Clara, CA95052
Phone/Fax(408)244-5358

Volume 1, Issue 6 $3.00

The "International" OS9 Underground®

Magazine Dedicated to OS9/OSK Users Everywhere!



Also:
- **Header Declarations**
- **Speed Disk**
- **Chaining in C**
- **C Declaration *Contest!***

SYSTEM IV

The 68000 Computer serving customers world-wide

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

Prices start at $999.00 with Professional OS-9

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

G-WINDOWS

for the SYSTEM IV and PT68K4/2

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

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

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

Copy and Paste between windows.

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

Runs existing OSK software without modification.

Number of windows and processes limited only by your memory.

Includes GIF viewer.

Includes G-VIEW demo.

G-WINDOWS with DESKTOP	$199.00
G-WINDOWS Developer's Pak	$299.00
Order both for	$449.00

OS-9/68000 SOFTWARE

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

QUICK ED - Screen Editor and Text Formatter	CALC-9 - Spreadsheet
VED ENHANCED - Text Editor	VPRINT - Print Formatter (for VED)
SCULPTOR - Development and Run-Time Systems	M6809 - OS-9 6809 Emulator/Interpreter
FLEXELINT V4.00 - The C Source Code Checker	DISASM_OS9 - OS-9/68K Dissassembler
WINDOWS - C Source Code Windowing Library	PROFILE - User State Program Profiler
IMP - Intelligent Make Program	PAN UTILITIES

delmar co

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

ColorSystems

Quality OS-9 Software for the CoCo3 and the MM/1 from IMS

CoCo3 Software	MM/1 Software
Variations of Solitaire $34.95	**Variations of Solitaire $49.95**
Variations Included: Pyramid, Klondike, Spider, Poker, and Canfield	Variations Included: Pyramid, Klondike, Spider, Poker, and Canfield
OS-9 Game Pack $34.95	**OS-K Game Pack $49.95**
Package Include: CoCothello, CoCoYahtzee, KnightsBridge, Minefield and Sea Battle	Package Include: Flip It, Dice Poker, KnightsBridge, Minefield and Sea Battle
WPShel $22.00	Coming Soon for the MM/1: Super Label Printer, X10 Master Control Program and Much More!
(Sorry! ColorSystems no longer carries MVBanner!)	

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

Offical Member of the Interactive Media Systems Developers Association

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

ColorSystems
P. O. Box 540
Castle Hayne, NC 28429
For Additional Information call at (919) 675-1706
North Carolina residents please include 6% Sales Tax

Write or call for a FREE copy of our Catalog!

Do you have a 'Fat Cat' in your house? Send us a picture and we'll print it...(Fat Dogs, Fat Birds, Fat Goldfish, are welcome too!) Yup, it's time for the first annual Fat Cat Contest. Please send pictures c/o Fat Cat Contest.

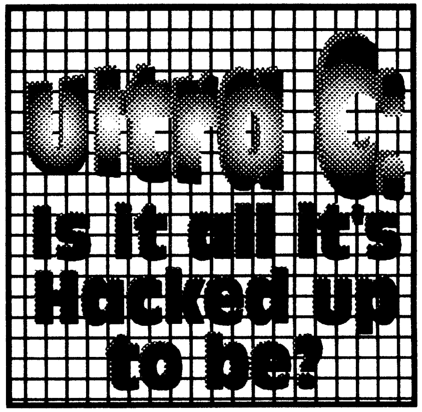
Next Month...
CHICAGO COCOFEST
Report!

Advertiser's Index

Company	Page
Delmar	IFC
Computer Design Services	7, 33
GaleForce Enterprises	10,12,14,,24
OS9 Underground	20, 40
JWT Enterprises	25
Canaware	27
Peripheral Technology	33
BlackHawk	33
Farna Systems	34
09-Online	38
Bob van der Poel Software	39
Peripheral Technology	39
AniMajik Productions	43
ColorSystems	IBC
ARK Systems	Back Cover

The "International" OS9 Underground® Magazine

CONTENTS
Issue Number 6



The Underground Staff

Editor/Publisher:
Alan Sheltra

Assistant Editors:
Jim Vestal
Steve Secord

Contributing Editors:
Scott Griepentrog
Leonard Cassady
Bob van der Poel
Paul Pollock
Mark Griffith
Wes Gale
Andy DePue

Art and Typesetting:
Alan Sheltra

Fat Cat Publications and The "International" OS9 Underground and it's logotypes are registersed trademarks. Subscription rates are $18.00 for 12 Issues. Single or back issues are available at the cover price (See elsewhere for a listing of back issues). Fat Cat Publications is located at 4650 Cahuenga Blvd., Ste #7, Toluca Lake, CA 91602 • (818) 761-4135

Mail Room

Letters to the Editor

Help is an Underground Away...

Dear Editor:

(Received from Delphi)

First off, I want to thank you for the great support you have shown this community since The Underground first 'opened it's doors'. Having another NEW magazine to breathe life in the Coco/OS9/68k community is just another good reason to look forward to the future we have. Also I would like to thank you for helping me solve a problem that millions of messages haven't been able to do. You got my C-Compiler running!

Jim Vestal's article "C for Beginners" helped me set up my C compiler in no time with those two small modifications. If I only knew it would be that simple ☺. Even tho I am currently not into C programming (I don't plan on learning it for some time to come as Basic is keeping me busy), it sure is nice to be able to actually compile some source code for once! I wouldn't mind seeing a series made from that article actually. I do have one suggestion tho. I was surprised to find that Jim never brought up the method of just using a Disk Editor such as DED or Kwikzap to do the job. That is how I did it as it seemed much simpler than the other methods. Here is how I did it with DED. I typed "Ded ccl" with cc1 of course being in the current working directory.

Then hit the up arrow 14 times. Position the cursor over where it says /dd/lib/cstart <the second to last line on the page> and hit the <Break> key to toggle ascii edit mode. Then position the blinking cursor over the '1' in the '/d1' and simply type 'd' or whatever drive you choose to use. I use my harddrive(/h0) as /dd so that is what I typed. Then simply press <Enter> to exit the edit mode. Next press 'V' at the command line to verify it then you are all set with the cc1 program. You can then press 'Q' to quit at the command line. Now with CPREP it is pretty much the same. Enter at command line "DED c.prep" then at the display hit the up arrow 19 times then hit 'e' to enter the edit mode. Then hit the down arrow 5 times and you will be at the line where the "/d1" is located. Hit <Break> to toggle the Ascii edit mode then hit the right arrow untill you position the cursor over the '1' and type 'd' or whatever drive number you prefer to use. Then press

C Langauge Declaration Contest

Decipher the following declarations in C and identify the declaration type cast. It is not necessary to include the steps you've used to arrive at the answer.

Hint: *Start with the variable name by itself and add each part of the declaration, starting with operators that are closest to the variable name as illustrated in the example below.*

Example:

Problem: char *a[];

[solution steps]

1). a[] is an array.
2). *a[] is an array of pointers.
3). char *a[] is an array of pointers to chars.

Answer: An array of pointers to chars.

Good luck and watch the parentheses which will change the precedence order.

Complex Declaration Contest Problems

Prob #

1). int a; ____________
2). int *a; ____________
3). int a[]; ____________
4). int a(); ____________
5). int **aa; ____________
6). int (*aa)[]; ____________
7). int (*aa)(); ____________
8). int *aa[]; ____________
9). int aa[][]; ____________
10). int ***aaa; ____________
11). int (**aaa)[]; ____________
12). int *(*aaa)[]; ____________
13). int (*aaa)[][]; ____________
14). int **aaa[]; ____________
15). int (*aaa[])[]; ____________
16). int *aaa[][]; ____________
17). int aaa[][][]; ____________
18). int *aa(); ____________
19). int (**aaa)(); ____________
20). int *(*aaa)(); ____________
21). int *aaa[](); ____________
22). int (*aaa[])(); ____________
23). int **aaa(); ____________
24). int (*aaa())[]; ____________
25). int (*aaa())(); ____________

Good Luck!

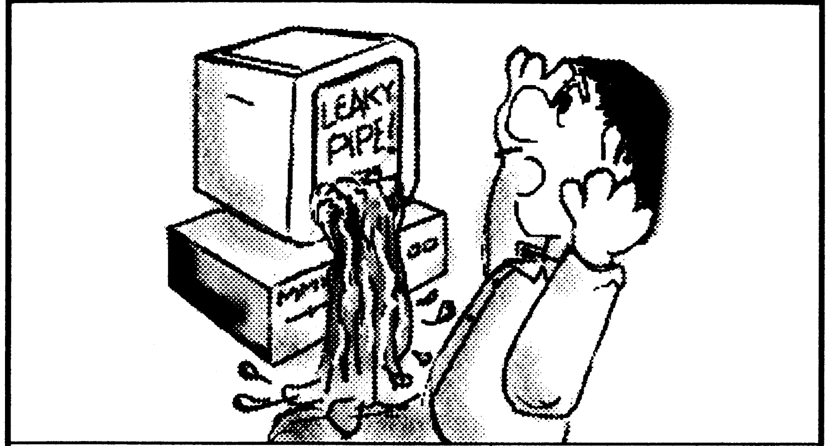
(C Declaration Contest deviously devised by Leonard Cassady)

C Language Declaration Contest...

Follow the directions and fill out the answers on the next page. Send your answers to the OS9 Underground Magazine before June 15th. You may send a photocopy if you like. First person with the correct answers to all the questions will receive a $10.00 check.

Underground Staff are not eligable to enter
Answers must be postmarked before June 15th to qualify
Winner will be announced in an upcoming issue.
Please include, your Name, Address and Phone number.
Winner will be contacted by phone on June 16th.
Send your entry to: The OS9 Underground
C Costest
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

SHELL GAME BY ALAN SHELTRA



Smedley has discovered a new error in his strange use of pipes, requiring the services of a plumber as well as a computer tech.

Enter to exit the edit mode and hit 'V' to verify, like you did earlier. Then just hit *Q* to exit and you are all set. It's that simple! Now keep in mind, it still hard code the Defs and Lib directory after whatever drive number you choose so if you want a different pathname, you will have to change the whole pathname instead of just the drive number while in DED, but that isn't any harder so don't worry. It's just a little more typing☺. Before I 'go', I would just like to recommend that you have someone write such an article for Both the Pascal Compiler and the level 1 Assembler since that is something that stumps many people, especially the Pascal compiler which many people feel hopeless in getting it set up. Also a series of articles profiling the BBS' that currently serve our community <1 bbs an issue> would be a nice touch, so that attention can be given to those that work hard with their boards to help the community ☺. Thanks again!

Chris Perrualt

Thank you, Chris, for the kind words. An upcoming issue will feature just what you are looking for on Pascal, Basic09 and C. BBS authors are always welcome to tell us a about their systems here. (See below for address to contact the OS9 Underground.) - Editor

Loves a Fight

I have been really enjoying the auguement between Paul Pollock and Ed Gresick about the pros and Cons of 68K. Will you continue this thread?

I, for one, an in Ed's corner about this having had an a 68K, and realizing the potential of the OS and really felt Paul was talking in "*world of a few years ago*".

Jim Sutemeier

In a word, Nope!! I feel the continuation of this thread would only continue to hurt the community and do nothing but serve to confuse and spread misinformation.

I will also have to side with Ed on this one too. Paul, I'm afraid stuck his foot in his mouth with his rantings, of which I as editor, caught most of the heat. So, no, I will be VERY careful about the content that goes into this magazine from now on.

Jim Sutemeier, my good friend, has also stepped down as associate editor. Jim Vestal has is filling that role as the new Assistant Editor of this Magazine. -Editor

How to leave your feedback...

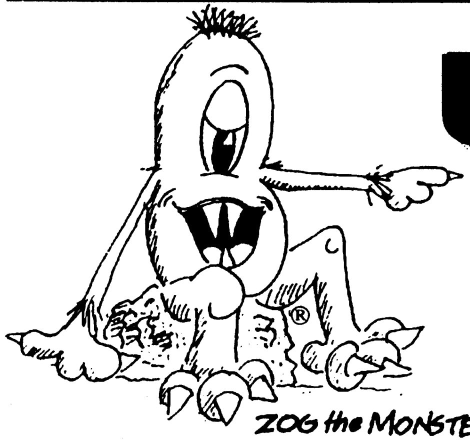
Letters to the editor may be addressed to:

OS9 Underground
Letter to Editor
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

or to either of the following Email addresses:

"zog!sysop@abode.ttank.com"
"JSUTEMEIER@DELPHI.com"

<EOF>



Under it All

Return from Down Under ...

It's been a **long** absence but the Underground is back... and back to stay. Many of you may have not been aware, but the publisher (namely me) had serious finnancial problems for several long months. That compounded with a recent death in the family, brought me to an all time low. My choice was simple. I had no choice but to put the magazine on hold.

Over the last few months, I have been working hard to get back on my feet. I've finally gotten to a point where I can now put the Underground out once again, and on a regular basis.

I must apologize to all my readership for these incredible delays you've had to put up with. and thank you all for bearing with me.

The MOTD, of which I was also the editor, has also been passed on to a new editor (my term of office was up April 1st 1993). This of course means, I have more time to dedicate to this magazine, instead of deviding my time between 2 publications.

Most of the articles in thisissue are about 3 to 6 months old, but, that's about average for most publications anyway. In the next few issues, articles will be much more timely and the turn-around time from author to print will be 1 to 3 months max.

Changes in Staff

There are a few changes in the Underground Staff. **Jim Sutemeier**, who has served as Associate Editor steps down. **Jim Vestal**, former submissions editor will now serve as Assistant Editor. Thanks to both "Jim's" for their dedication, time and effort to this magazine.

There may be a few more changes shortly which I will announce in the next issue.

Next month, things will be back on track once again. Look for a Report on this last May's Chicago CoCoFest. Things are looking up for the World of OS9!

<EOF>

```
pshs x              pass it
lbsr printf         call printf
leas 8,s            strip off parameters

*}
leas 22,s           clean up "main's" local variables
puls u,pc           retore our U register and exit the program
_1 equ -98          this is the C compiler's assesment of how much stack
*                   space is needed for our program
*                   this can be set by one of the linker options

_2 fcc "This is a test"
fcb $0

_3 fcc "%s"         this is our printf 'format' string.
fcb $d              for the first call to printf
fcc ""
fcb $0

_4 fcc "NUM = %d, DIGIT = %d, ERRNO = %d"
fcb $d
fcc ""              as well as this one, for our
fcb $0              second call to printf

endsect
```

If you read the comments and follow what the code is doing you will see how parameters are placed on the stack for the routine you want to call. It will also show you how local variables are 'created' and accessed. I think this is enough for now. Next time I will have an actual routine that does something that you can use.

Wes Gale Delphi WESGALE FIDO Net 1:153/912 or (604)589-5545 24 Hrs. STG Net sysop@kzin or (604)589-1660 5PM - 8AM Pacific Standard Time UUCP gale@f752.questor.wimsey.bc.ca

<EOF>

Keyboard Special! LIMITED QUANTITIES!

- Auto-Switchable for AT/XT
- Small-Sized Keyboard, for Computing on-the-Go
- Only 12.25" x 5.75" x 1.125", Fits in Briefcase
- Works with MM/1, TC70, KixOXO, IBM Compats
- Function Keys F1 thru F10

CALL (818) 761-4135

Personal Check or M.O.
Only $59.96
Price Includes S&H

Available from:
AniMajik Productions
4650 Cahuenga Blvd., Ste #7
Toluca Lake, Ca 91602

```
*               to know where the lowest point the stack can go is.

 ldd #_1                    get how much room we need for our local variables.
lbsr _stkcheck              and check if we have room
*                           the routine _stkcheck is a routine in CLIB.L for checking
*                           the stack to make sure we don't go over any boundaries
*  char str[20];
*  int digit;
leas -22,s                  this moves the stack pointer down to make room for our
*                           local variables.  20 bytes for our 'string', and 2 for the
*                           integer.  The last variable specified will be where the
*                           stack pointer is set to.  In this case :
*                                   'digit'  =   0,s  (length is 2 bytes)
*                                   'str'    =   2,s  (length is 20 bytes)

* parameters are placed on the stack in reverse order making
* the stack pointer equal to the first parameter upon entry of
* the routine you call.
*  ie.
*               parameter 1 = pointer to var 'text'      =    0,s
*               parameter 2 = pointer to "this is a test"  =   2,s *
*  strcpy(text,"This is a test");
leax _2,pcr                 this gets the pointer to our string
pshs x                      put it on the stack - our second parameter
leax text,y                 get the pointer for the destination variable
pshs x                      pass it as the first parameter
lbsr strcpy                 call the routine to copy the string
leas 4,s                    clean up the stack - deallocate what we pushed

*  printf("%s\n",text);
leax text,y                 get the pointer to our second parameter
pshs x                      pass it to printf
leax _3,pcr                 point to our 'format' string for printf
pshs x                      and pass that too
lbsr printf                 call printf
leas 4,s                    clean up the stack


*
*  num=5;
ldd #5                      make 'num' = 5
std num,y                   offset from Y to our global variable

*  digit=num;
ldd num,y                   grab 'num'
std 0,s                     and make 'digit' equal to it.

*
*  printf("NUM = %d, DIGIT = %d, ERRNO = %d\n",num,digit,errno);
ldd errno,y                 grab last parameter pointer
pshs d                      pass it
ldd 2,s                     get the next parameter pointer
pshs d                       pass it
ldd num,y                   grab the next parameter
pshs d                      pass it leax _4,pcr      grab the first parameter pointer
```

NEW 68020 COMPUTER

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

CD68X20-25, 0K RAM	$699.00
1MBx9 60ns SIMM Module	$ 59.00
Professional OS9/020 V2.4	$499.00
SCULTOR 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

Ultra C: Is it all it's Hacked up to be?

by Scott t. Griepentrog

Overview

In late '92, Microware started shipping its first C compiler (for OS9 68k & OS-9000) that meets the ANSI standard. But as usual, Microware did their own thing. Rather than just creating a program that compiles code in the usual methods, they turned the process upside down. The result is a compiler that should be able to make optimized code good enough to turn PC users green with envy. This article takes a look at the obvious question: did they succeed?

Background

In order to understand what makes the Ultra C Compiler (hereafter 'UCC') different, it is first necessary to review how ordinary compilers function. The command "cc source.c", for example, actually calls up several different steps to process the code. The source file is first converted to assembly source (source.a), then passed through an assembler to create a relocatable object form (source.r). The .r file is then linked with the cstart.r (which contains the c startup code that calls the main() function), and then any other functions still undefined are (hopefully) located in a library and linked in. If everything goes okay, the linker outputs an executable module and the cc command completes.

A variation of this procedure occurs when more then one source file is compiled together. Each of the source files specified is compiled, one at a time, through the assembly to the .r form. Only after all the source files have been individually converted to object files are they linked together to form an executable module. In each case where a function is called from one object file that is contained in another, the relative offset in the branch instruction is set by the linker to reflect the proper location in the executable module. The same is done for global variables that are linked between different source files as well. The drawback to this method of compiling is that every time a call to a function is made, the same procedure must be used. Each parameter must be put in certain registers or on the stack in a certain way for functions to find them. This means that even if you call a function like:

```
abs(x)
int x;
{
        if (x<0)
                return(-x);
        return(x);
}
```

The entire procedure for calling a function must be included in the code. The current stack pointer is saved, then stack space is allocated to pass the x variable, then the function is called, and the stack is restored. If the -s (no stack checking) option is not used

```
/* test.c */
/* this is a do-nothing C program to demonstrate how to write assembly library routines
   for the MW C compiler for OS9 LII */
char text[80];
 int num;
main()
{
        char str[20];
        int digit;

        strcpy(text,"This is a test");
        printf("%s\n",text);

        num=5;
        digit=num;

        printf("NUM = %d, DIGIT = %d, ERRNO = %d\n",num,digit,errno);
}
```

The variable 'text', and 'num' are both global. Both 'str' and 'digit' are local to the routine 'main'. 'errno' is a global variable contained in 'clib.l' for returning error numbers when an error occurs in some of the standard C library routines. Global variables can be access from anywhere, local variables are made temporarily during execution of the routine they are contained in, on the stack. They can only be accessed from within the routine they are contained in. Below is the assembly language code that the C compiler created from our above C program.

This was done using two options in 'cc' - I asked the compiler to stop at the assembly language stage, and include the C code as comments:

```
        cc -a test.c
```

```
 psect test_c,0,0,0,0,0
 nam test_c
* global variables are allocated when you execute your program
* The C compiler uses the Y register as a data pointer, all references to
*   global variables are done as an offset from Y

* char text[80];
 vsect
text: rmb 80      this is our global variable 'text'
endsect

* int num;
vsect
num: rmb 2        this is our global variable 'num'
endsect

*
* main()
* {
ttl main
main:             global labels are set by placing the ':' after it
pshs u            save the U register - it is used by the C compiler
```

BUILDING YOUR OWN C LIBRARIES

by Wes Gale

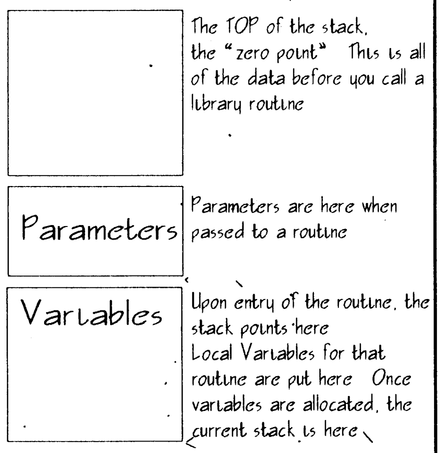
Have you ever wanted to write your own C library routines? Maybe you even wanted to write them in assembly? I would like to share with you a few things I have learned about doing just this, which is something I had trouble with when I was first learning how to do it. I hope to help portray this information in a way that is a little easier to grasp that the information I had to go on when I was learning.

Assembly language was the second language I learned, the first was BASIC. I learned BASIC under RSDOS and wanted more speed, so I learned assembly. Once I moved to OS9 I learned C. Knowing assembly helped making learning C a lot easier, a lot of things in the C language are based on how assembly language works. This is one reason C is so easy to use for low level coding without resorting to assembly.

Our little machine here is not exactly a speed demon, so after learning C a little bit, I started trying to make library routines that were decently small, and fast. In these articles I will show you how global and local variables are accessed, how to pass parameters to your routines, and how to return information back to your program calling the routine. I will use a few of the library routines I have written for examples, maybe you will find them useful in your own programs.

Some of my routines depend on having the Carl Kreider C libraries. If you do not have these, you should attempt to get them. They will save you time, and make your code a little bit smaller, and faster - many of his routines are written in assembly. If forcing you to have these presents a problem, let me know, I will try and work around that in the future.

The stack is used for all temporary variables and for parameter passing. The stack starts at a certain point in you program's memory and works its way down. The following diagram illustrates where these are placed in memory. The top of the stack is the higher memory address. When something is 'pushed' on the stack, it drops the stack pointer down (if there is room) and places the necessary data in the memory is just allocated.


The TOP of the stack, the "zero point" This is all of the data before you call a library routine
Parameters
Parameters are here when passed to a routine
Variables
Upon entry of the routine, the stack points here
Local Variables for that routine are put here Once variables are allocated, the current stack is here

I am beginning with a very simple C program that has a couple of glabal variables, and local ones. It also makes a few calls to the standard C library routines to demonstrate how parameters are passed.

when the code is compiled, the function called will then call a library routine to make sure it has enough stack for local (automatic) variables. Both compilers (and only more recent PC compilers) use registers to pass the first two arguments, which saves a certain amount of stack overhead (but only if -s is used). But to do the comparison, jump, and a negate (three assembly operations in abs()), there can be a lot of overhead (as much as 10 operations). All because the code is made easier to read by making a common operation into a function. And if this function where called constantly, that extra overhead can really add up!

Better Method

For a compiler to produce assembly output near to what could be done by hand would require knowledge of the entire C source at one time, libraries and everything, and a routine that could decide how to make the best use of the registers available. Throw in some extra features to clean up sloppy coding, and you've got one heck of a compiler. That is basically what UCC does. It first converts each source.c file to an I-Code form source.i (somewhat like Pascal p-code). Then all the .i files (still in near-source form) are linked together, along with I-Code libraries, into a single .i file, which passes through an optimizer. After that, the .i file is converted to assembly, which is again optimized and assembled into the familiar .r file. Only this time the entire program is contained in the one .r file, which can then be linked with any older libraries still in .r form to make an executable.
One of the advantages of the I-Code format is that different 'front end' converters could be used to interpret different languages into I-Code. For example, a C++, Basic09, or Pascal front end would add another language using the same optimization capability and libraries. Currently, only a C front end exists, but it wouldn't surprise me if Microware is already working on another. Another advantage of using this I-Code standard in the middle is that different 'back end' converters can be used (these convert the I-Code to assembly for a particular target processor). For example, back ends for creating '030 and CPU32 specific code come with the UCC package, in addition to the default 68000 one.

I-Code Optimization

In the middle is the I-Code optimizer, which implements quite a list of methods for improving the code. The first thing it looks for is places where variables are set to a constant once and never changed. In these cases it can replace the variable itself with the constant.

Next the optimizer keeps track of how many times a variable is used and where so that it can make the best use of registers and eliminate cases where they are not used at all. It also looks for ways of simplifying calculations such as the use of multiple constants and multiple occurrences of the same calculation. Next, it looks for sections of code that are duplicated or repeated and changes the flow to either remove the duplications (thus making the code smaller) or moving the repeated section outside of the loop (thus making the code faster). Finally, it will inline functions whenever it can. For example, the abs() function above is short enough that it would actually be replicated each time it was used. Some optimizations have an execution speed vs. code size tradeoff, which can be set as a ratio on the command line.

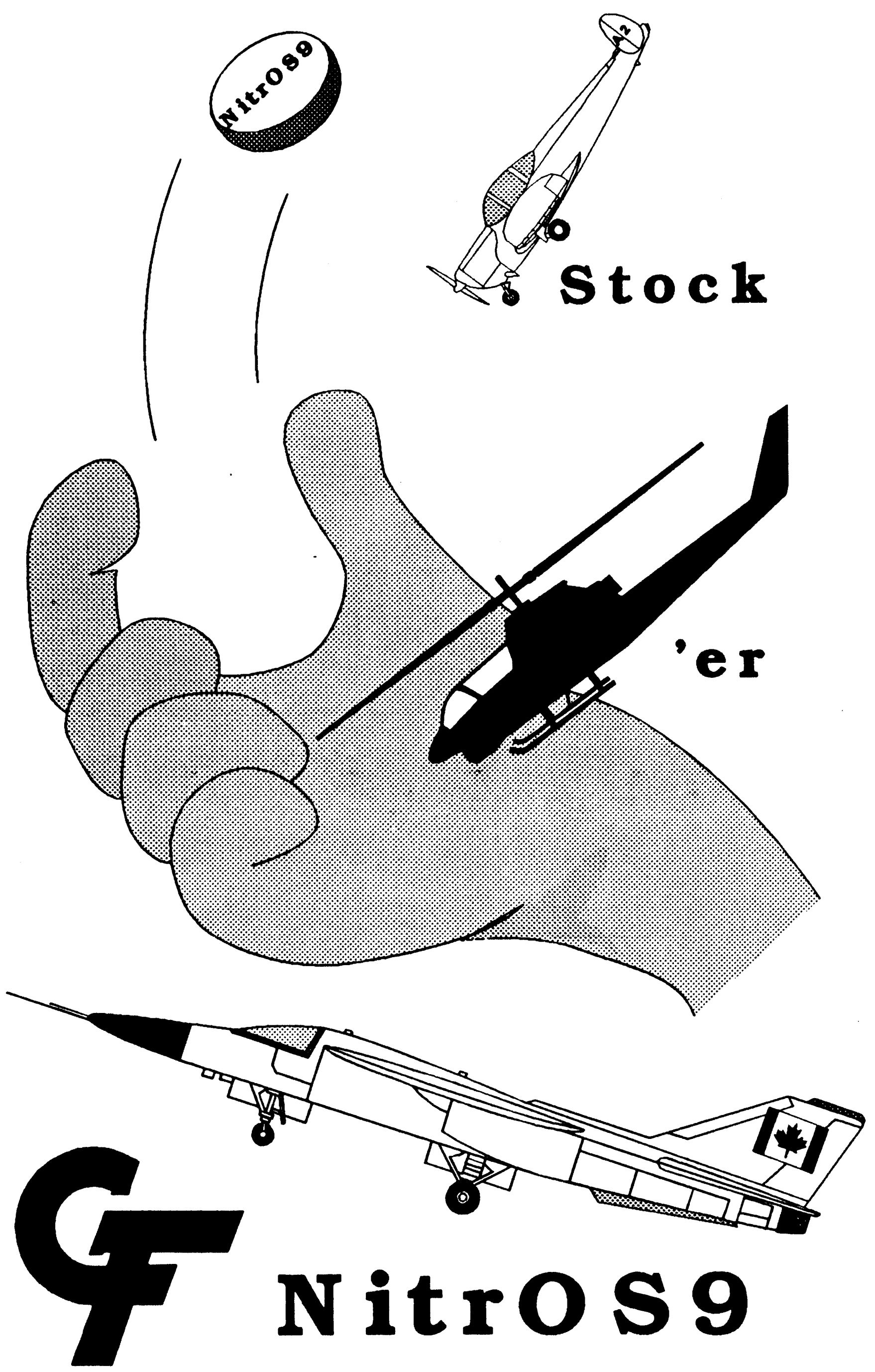
Code Changes

There are a number of things that have been changed between the old C compiler and UCC. The first, and most aggravating, is the removal of #asm and #endasm (for ANSI compatibility). In place is the new _asm() pseudo-function, which has the drawback of not being allowed inside of functions.

Microware supplies a deasm program to handle the conversion, but it too only handles assembly outside of functions. I have found that the easiest way to take care of #asm's inside functions is to split the function into a separate .c file and compile it to assembly with the old compiler.

Another change is the missing OSK define. It has been changed to _OSK, which, I was surprised to learn, was also defined in the old compiler. The quick solution is to change all ifdef's from OSK to _OSK and your code will still work with either compiler. To tell which, check for the new _UCC define.

Other additions are the volatile and const typecasts. To prevent the compiler from optimizing a variable incorrectly (such as one modified by a signal service routine),



Vprint/68000 Text Formatter

The latest addition to our OS-9/68000 product line is the most powerful text formatter available. Vprint will work with *any* printer from files produced by your favorite editor. Proportional character sets are fully supported as well as most of the special features newer printers have. It even works with laser printers.

Standard features include margins settings, indents, headers, footers, etc. Advanced features include multiple column output, repeats, powerful macros with optional parameter passing, internal number registers with many output formats, true footnotes, automatic indexing and table of contents generation, future event testing...

And if that's not enough, Vprint has a complete string manipulation language It supports documentation via change bars, marginal notes and boxed sidebars; and permits I/O redirection to and from pipeline.

Send for a FREE sample printout demonstration some of the many advanced features!

Vprint comes with a 100 page manual and loads of sample files. It can be configured by any user to any printer. Vprint 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
Porthill, ID, USA 83853 or
USA 83853**

**P.O. Box 57
Wynndel, BC
Canada V0B 2N0**

Phone (604) 866-5772

SCULPTOR FOR OS9

V1.14:6 of the well-known development software package, SCULPTOR, is available for all OS9 computers. Two high-level languages plus a complete set of support programs, including utilities, are included with a detailed manual and program disk. OS9 (any version) and one floppy drive are required; a hard disk is recommended. SCULPTOR is sold "as is" but may be returned within 5 days if you are not satisfied. Quantities are limited.

SCULPTOR V1.14:6 **$79.00 + $7.00 UPS Ground**

*COMPUTER DESIGN SERVICES
2550 Sandy Plains Rd. Ste. 320-234
Marietta, GA 30066
404/973-2170*

```
PRINT "Press any key"
GET #0,key
PRINT CHR$(12)
PRINT "Border Colors"
PRINT "——— ———"
PRINT
FOR counter:=1 TO 8
RUN color("border",code(counter))
PRINT "This is "; code(counter)
FOR time_delay:=1 TO 9000
NEXT time_delay
NEXT counter
RUN color("border","black")
END
DATA "white","black","red","green","yellow","blue","purple","cyan"
```

<EOF>

09-Online Systems

Spring/Summer Modem Sale

US Robotics Sportster 14,400 Modem External Version
CCITT V.32 bis with V.42 bis throughput up to 57,600 bps

Special sale price: $250.00 (add $5 for UPS shipping)

Send your paid order (check/money order), or inquiries to:

09-Online Systems
c/o Jim Vestal
221 E. 17th #31
Marysville, CA 95901

Ask to be placed on mailing list for a free shareware catalog.

make the compiler allocate constant variables or arrays (that aren't changed) in the program space (instead of making a copy in data space), use the const qualifier.

Compiling Options

There is way too many options to this compiler to go into them here, but certain ones are very interesting. For example, the -mode={ucc | compat | c89} option actually switches the meaning of all the other options. By selecting compat, you can use all the familiar options of the old compiler and still have the benefits of UCC. The c89 mode provides ANSI standard options, and the ucc mode gives you full control of the compiler. All the following options are for the ucc mode. The options I have been using are -r (no stack check), -j (include I-Code libraries), -o=7 optimization 0-7), -t=2 (time/speed optimization at 2:1), and -tp=68kc (compile for 68k w/word code references). I also add -ill=/dd/ucc/lib/sys_clib.il to make the compiler recognize the 'old' library functions. This is necessary because the default libraries have either _os_ or _os9_ in front of many of the functions. For example, write() is now _os_write()

Put to the Test

To find out just how much better the UCC compiler is, I took a few programs I had just lying around and timed how long they took to compile and run. All three programs just crunch data in memory (no I/O), all times are in seconds, sizes in bytes, and all tests were performed on a 33Mhz 68030 Heurikon V3D. The UCC versions where compiled with the options -mode=ucc -r -j -o=7 -t=2 -tp=68kc, except for #3 which also needed -beb=300k because of it's size.

	OLD	UCC	Change
Program #1 (46 lines)			
Compile time:	11	343	31 times longer
Execution time:	45	22	half the time
Code size:	13694	19018	39% larger
Program #2 (280 lines)			
Compile time:	15	351	19.5 times longer
Execution time:	1185	1140	less than 4% faster
Code size:	14802	20218	over twice the size
Program #3 (6439 lines)			
Compile time:	149	1332	almost 9 times longer
Execution time:	1.908	1.815	5% faster
Code size:	57126	60646	6% larger

The first program consists of a simple repeated loop with several sub-functions (including abs()). UCC is able to inline several of the functions, resulting in a considerable speed increase. But what is disappointing is that it does not improve complex programs by as much. Having looked through the assembly code generated by program #2, I am quite certain I could speed it up by at least 25% if I took the time to rewrite it in assembly.

Installation Procedure

Before installing UCC, you need a 68k OS-9 machine with at least 2 meg of ram and 5 meg free on a hard drive. The faster the machine and the larger the memory and the bigger the hard drive the better. UCC comes on seven 3-1/2" or 5-1/4" 720k (OS-9 'Universal' format) disks. It comes with it's own install program. In fact, you MUST install it using their program as the disks are in some sort of tar format. Now, I realize that this installation program is a generic routine that can and is used for different packages, but it has some serious problems when used to install UCC. On a package of this size (UCC is a little over 4 Meg) and complexity (scripts are used to copy existing defs and lib directories), this half-baked install program just doesn't cut it.

NITROS9

Version 1.07
OS9 Level II expediter

Give OS9 Level II what it really needs,

A little speed...

NITROS9 offers :

- **-Faster Interrupt handling**
- **-Optimized System calls**
- **-Faster graphics**
- **-Smoother multitasking**
- **-Faster text**
- **-Faster floppy/hard disk I/O**

Upgrades to NITROS9 version 1.xx will be made available for FREE through Delphi and Compuserve and other BBS'.

If you do not have access to these online services, upgrades can be ordered for only a small handling fee.

NTIROS9 requires - OS9 Level II
- an HD63B09E installed in your COCO III.

NITROS9 software only	$34.50
NITROS9 Kit	$49.50

- comes with complete installation instructions plus necessary hardware (minimal soldering experience required)

Experience GALE FORCE speed!

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

Send cheque or money order to :
Gale Force Enterprises
P.O. Box 631, Surrey, BC,
Canada, V3T 5L9

(604) 589-1660

8 AM - 5PM PST (voice)
5 PM - 8 AM PST (support BBS)

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

```
NEXT I
(* change the value of word to the new value, passed to the calling program *)
word:=new
END
PROCEDURE demo
(* A demo program for use with the procedure OS-9 Color *)
(* By Timothy J. Mohr *)
DIM code(8):STRING[15]
DIM control:STRING[15]
DIM counter:INTEGER
DIM time_delay:INTEGER
DIM key:STRING[1]

(* read color names into an array *)
FOR counter:=1 TO 8
READ code(counter)
NEXT counter

counter:=0

(* clear the screen *)
PRINT CHR$(12)
PRINT "Foreground Colors"
PRINT "————— ———"
PRINT
FOR counter:=1 TO 8
RUN color("foreground",code(counter))
PRINT "This is "; code(counter)
NEXT counter
PRINT
RUN color("foreground","white")
PRINT "Press any key"
GET #0,key
PRINT CHR$(12)
PRINT "Background Colors"
PRINT "————— ———"
PRINT
RUN color("foreground","black")
FOR counter:=1 TO 8
RUN color("background",code(counter))
PRINT "This is "; code(counter)
NEXT counter
RUN color("foreground","white")
RUN color("background","black")
PRINT
```

```
IF code="black" THEN
color:=2
ENDIF
IF code="red" THEN
color:=4
ENDIF
IF code="green" THEN
color:=3
ENDIF
IF code="yellow" THEN
color:=5
ENDIF
IF code="blue" THEN
color:=1
ENDIF
IF code="purple" THEN
color:=6
ENDIF
IF code="cyan" THEN
color:=7
ENDIF
IF command<>$FF AND color<>$FF THEN
seq:=CHR$(esc)+CHR$(command)+CHR$(color)
PUT #1,seq
ENDIF
END

PROCEDURE to_lower
(* to_lower procedure by Jim Vestal and Tim Mohr, this can be used in any *)
(* to convert the passed string to all lowercase *)
PARAM word:STRING \(* word is passed from calling program *)
(* new is the "new" word and should be all lowercase *)
DIM new:STRING
DIM I:INTEGER \(* Loop counter *)

new:=""
I:=0

(* change all uppercase letters to the lowercase counterpart *)
FOR I:=1 TO LEN(word)
IF MID$(word,I,1)>="A" AND MID$(word,I,1)<="Z" THEN
new:=new+CHR$(ASC(MID$(word,I,1))+32)
ELSE
new:=new+MID$(word,I,1)
ENDIF
```

First, the installation takes a full 45 minutes on an '030. That's over six minutes per disk - and it doesn't even take a minute to copy in a single floppy! Using dsave would take a fraction of the time and would still only use 7 disks. Second, you must install it straight to /dd or the scripts fail. Is this specified in the installation instructions? No. Like most people who like to keep stuff on their hard drive somewhat organized into subdirectories, I first tried to install it to /dd/UCC and got all kinds of error messages.
Although the scripts attempt to make a copy of your old compiler's modules and files (in case you want to go back to it), it will mess them up completely if everything doesn't go right on the first try. And since what the scripts are doing is not echoed to the screen, it took me a bit to trace down what happened the first time I tried installing it. To put it nicely, this installation program bites.
For the poor unfortunates who are faced with performing this installation, I provide the following procedure. This assumes that the installation disk is /d0 and you wish to place UCC in its own directory (to make it easier to switch back and forth). To install permanently, skip straight to running /d0/install and enter /dd as the destination directory instead.

```
makdir /dd/UCC
makdir /dd/UCC/DEFS
makdir /dd/UCC/LIB
chd /dd/defs;dsave -e /dd/ucc/defs
chd /dd/lib;dsave -e /dd/ucc/lib
chd /d0;install

(enter destination device) /dd/ucc
(enter pathlist to target installation directory) /dd/ucc
(ignore errors)
(select option by number) 2 (overwrite pre-existing files)
(are you sure) y
(ready for volume 2) go
(... repeat for 3-6 ...)
(ready for volume 7) go

(this sets the modules with the same name as non-sticky to prevent conflict)
chd /dd/ucc/cmds
attr -w cc r68 l68 make fixmod rdump
fixmod -ua=8001 cc r68 l68 make fixmod rdump
chd /dd/cmds
attr -w cc r68 l68 make fixmod rdump
fixmod -ua=8001 cc r68 l68 make fixmod rdump

(finally, edit the /dd/startup file and add the following lines:)
load -d /dd/ucc/cmds/csl
load -d /dd/ucc/cmds/bootobjs/fpu  (use fpu040 on a '040)
/dd/ucc/cmds/p2init fpu
```

To make it easy to select ucc over the old compiler, I have also come up with this quick C program that goes with my installation procedure. When run, it chains to a shell with the proper environment set to run the new compiler using the same commands (defaults to -mode=compat). This way you can just cc your old code just like before. You can also run make, but it will not link source together in I-Code form first.

KWIKGEN

Version 1.01

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

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

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

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

KwikGen requires OS9 Level I, or II. ***$24.95***

KWIKZAP

Version 1.2

- Display updating is instant
- 'Smart' verify command
- Work on file or stack
- Searching functions
- 100% assembly code
- Configurable environment
- Dynamic sector stack
- Allows editing of nibbles or half bytes
- Built in help - easy to use

KwikZap requires OS9 Level II. ***$24.95***

Experience GALE FORCE speed!

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

Send check or money order to :
Gale Force Enterprises
P.O. Box 631, Surrey, BC,
Canada, V3T 5L9

(604) 589-1660

8 AM - 5 PM PST (voice)
5PM - 8 AM (support BBS)

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

```
PROCEDURE color
(* OS-9 Color *)
(* Version 2.0 by Jim Vestal *)
(* Version 1.0 by by Timothy J. Mohr *)

(* control is the string "foreground","background", or "border" *)
(* code is the name of the default 8 colors, "white", "black", "red", "green", "yellow",
"blue", "purple", and "cyan" *)
PARAM control,code:STRING[15]
DIM new:STRING
DIM count:INTEGER
DIM seq:STRING[3]
DIM esc,fg,bg,bd,color:BYTE
DIM command:BYTE
seq=""
(* set values to maximum for error checking *)
color:=$FF
command:=$FF
esc:=$1B \(* escape display code *)
fg=$32 \(* foreground color display command *)
bg=$33 \(* background color display command *)
bd=$34 \(* border color display command *)

(* Convert control to lower case *)
RUN to_lower(control)

(* Compare control to "foreground", "background", and "border" commands *)
(* and set control to corresponding value *)
IF control="foreground" THEN
command:=fg
ENDIF
IF control="background" THEN
command=bg
ENDIF
IF control="border" THEN
command=bd
ENDIF

(* Convert code to lower case *)
RUN to_lower(code)

(* Compare code to valid color names, set color to corresponding color *)
IF code="white" THEN
color:=0
ENDIF
```

BASIC TRAINING

Structured Programming in Basic

by Jim Vestal

This month I present a procedure called OS-9 Color. This procedure requires Basic09 in an OS-9 Level 2 window on a CoCo 3, and it assumes the default palette values. It allows you to change the current foreground, background and border colors to any of the standard 8 colors by name. It may be executed as a stand alone program from the shell with the parameters of which color to be changed and the name of the color, or it may be used in any program in order to change your screen display color. For example, for shell plus users:

```
color foreground green
```

This command will change the foreground color to the color green (palette 3).

You may also call this routine from any basic program. For example:

```
run color("foreground","yellow")
run color("background","blue")
run color("border","purple")
```

These program lines will change the foreground color to yellow (palette 5), the background color to blue (palette 1) and the border color to purple (palette 6).

Reader Input Needed

I need input from you readers about what you would like to see in this column. Do you want small useful programs, or would you rather have tutorial discourse? Please send me email or write me at the following mailing address:

Jim Vestal
221 E. 17th #31
Marysville, CA 95901

You can send me email me on the following networks:

Internet:
sysop@narnia.citrus.sac.ca.us
StG network: SysOp@Narnia

RETURN OF THE RASCAN DIGITIZER!!!
The BEST CoCo Video Digitizer is now again available! 320x200 to 640x200, color scans with 512K and filters! Plugs into joystick ports. See Rainbow review, Dec. 1990, pg. 83. **INTRO SPECIAL:** Send $40 down and get a Rascan for **only $140!** Shipped from Australia; allow 8-10 weeks. Full payment only $135. Offer expires 30 Nov. 1990. **REGULAR PRICE: $175**

NEW ITEM: OS-9 Invoice/Inventory Manager. Print invoices by typing customer info then selecting part number from list! Complete with Basic-09 source. Multi-user , Level I or II. Requires Basic-09. $24.95

OTHER FARNA ITEMS
The CoCo Family Recorder - genealogy system, CC3, 80 column monitor, 2 drives- **$24.95**
OS-9 Quick Reference Guide- desktop reference!- **$7.95**
Patch OS-9 - 2 disks of most used patches&utilities, auto patches w/512K, minimum of 2 40T DS drives. **$5.00 w/QRG, $7.50 alone**
Little Black Book- address/phone database- **$7.50**
Video Tape Organizer- video library database- **$7.50**
LBB & VTO- both only $12.50!!
Keep-Trak- general accounting , CC 1,2,3- **$24.95**
Accounts Receivable for K-T- $10 ($20 alone)
Omega File- menu driven database, CC 1,2,3- **$24.95**
Programming the 6809- Assembly language book recommended by Motorola & Marty Goodman- **$22.95**
NOTE: OS-9 ONLY IF STATED. OTHERS DECB

FARNA Systems
904 2nd Avenue, Warner Robins, GA 31098
Phone 912-328-7859 **Add $1.50 S&H** (GA add 5% tax)

```
/* UCC.C - fork shell to use ucc */
/* StG 93/01/31                 */

char *malloc();
char *getenv();
int chain();

extern int errno;
char *env[16];
char *args[]={"shell",0};

set(e,v)
char *e,*v;
{
        static char **envp=env;
        char *new

        new=malloc(strlen(e)+strlen(v)+2);
        if (!new) exit(errno);
        strcpy(new,e);
        strcat(new,"=");
        strcat(new,"v);
        *envp++=new;
        return(0);
}

main()
{
        char *temp;

        set("PATH","/dd/ucc/cmds:/dd/cmds");
        set("CDEF","/dd/ucc/defs");
        set("CLIB","/dd/ucc/lib");
        set("PROMPT","UCC: ");

        if (temp=getenv("CC")) set("CC",temp);
        else set("CC","compat");

        if (temp=getenv("PORT")) set("PORT",temp);
        if (temp=getenv("HOME")) set("HOME",temp);
        if (temp=getenv("USER")) set("USER",temp);
        if (temp=getenv("TERM")) set("TERM",temp);

        os9exec(chain,*args,args,env,0,0);
}
```

Conclusion

Currently, considering the price, hardware requirements, time to compile, and lack of substantial return (i.e. faster code), UCC may not be a worthwhile acquisition quite yet. The next release is supposed to improve optimization and fix a few minor problems. It should be available in the Fall of '93, at which time I will run the same tests and post the results. In the meantime, the ANSI compatibility and improved source checking features may prove worthwhile to those with the necessary funds and equipment. - StG

Header Files and Function Declarations

by Bob van der Poel

So, here we are: starting a new year, folks continue to use their Color Computers, folks continue to get into OS9/68000, and authors (like me) continue to make mistakes... So, what's new?

One sharp reader questioned a comment I made in part II of my series on using termcap to build a simple menu program (UG, July/92, pg. 22). In that article I stated that I included the file "malloc.h" to keep the compiler happy. The reader felt that my explanation was lacking, plus he didn't agree with the file. So, let's discuss it a bit more.

C compilers make a lot of assumptions when compiling a program. One of them is that any function, unless it is specifically declared to do otherwise, returns an integer value. This assumption permits the following code to compile and work properly: *(See Example 1)*

(Example 1)

```
main()
   {
      int a,b,c;
      a=22;
      b=44;
      c=multiply(a,b);
      printf("%d times %d is%d\n",a,b,c);
   }
   multiply(a,b)
   int a,b;
   {
     return a*b;
   }
```

Of course, this isn't the most useful of programs—but it demonstrates the point.

Now, what happens when we use floats instead of integers? Apart from the obvious detail of changing all the "int" declarations to "float" and changing the "%d"s in the printf() to "%f" we need to do one other very important thing. We need to tell the compiler that multiply() is returning "float". If we don't an error will result—the function multiply() will convert the result of the "a*b" operation to an integer (possibly discarding part of the value)—we didn't tell it to return a float; this integer value will be assigned to "c".

For this to work properly we need to end up with code which looks something like: *(See Example 2)*

Advertise your Wares (Hard or Soft!) In the *OS9 Underground*
Ask for Ad Rate Card or call for more information
(818) 761-4135

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
Professional OS9	$299.00

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

BlackHawk Enterprises Official IMS Representative

P.O. Box 10552
Enid, OK 73706-0552
(405) 234-2347

Coming Soon:
Bloody Chess and OS9 Stats

OS-9/6809		OSK	
Ved	$25	Ved	$40
VPrint	$30	VPrint	$60
DML9 Mail List Mgr	$25	The Zapper	$30
Basic09 Subroutines	$25	CheckBook+	$35
MiniBanners	$20	BGFX	$40
CheckBook+	$25		

NOW IN STOCK: DATADEX! Free Form DataBase	Multi-User Version	$300
	Single-User Version	$130

DATADEX SPECIAL 10% OFF in November and December ONLY!

HARDWARE

MM/1 Extended Kit	$975	3 Meg Memory Upgrade	$120
MM/1 Personal Kit	$795	MM/1 Case/Supply	$125

Send SASE for PD Offerings and Complete Catalog by changinit to a NULL. Adding the extra space fixes this little bug in the OSK version of chain. The OS-9/6809 version is a little different. You may have to add a space at the BEGINNING of the parameter string. If your chain call does not work and you are chaining to a shell, play around with spaces at either end of the parameter string. However, don't have any spaces in there if you are directly calling an executeable module or it will fail.

All this is pretty simple so far. Under OSK, it becomes slightly more difficult. The OSK chain() works like this:

```
chain(modname, paramsize, paramptr, type, lang,datasize, prior)
char *modname, *paramptr;
int paramsize, type, lang, datasize, prior;
```

The only new parameter is "prior", with which you can specify the priority of the new process. Set it to zero if you want the chained process to run at the same priority as the calling process. Otherwise, the OS-9/6809 and OSK versions of chain() work the same.

The OSK chainc() function is the same as chain() with the addition of yet another parameter "pathent" to specify the number of opened paths to pass to the new process. Usually this is three, for stdin, stdout, and stderr.

Notice that there is no need to explicitly exit the calling programabove. Remember, chaining causes the currently running process to terminate. Although you are very careful in setting up your chain function call, it is possible that it might fail for some reason. In the above example, if you spelled the name of the program wrong, prnit_report instead of print_report, the chain would fail with a "file not found" error. However, the calling program is now no longer in memory and the error cannot be returned to it. How is one to know the chain failed? The answer is the parent process of the one making the call gets notified. Usually, you run a program from the shell which might in turn chain to another program. If that chain failed, the parent of the chaining program—the shell—would get the error notification and print it.

Under OS-9/6809, chain can be very useful when trying to write the "killer app" in the tiny 64K address space given to each program. By dividing up the application into a logical sequence of modules and then chaining to each of them as the program runs, you can create the illusion of a much larger program. Careful thought must go into this though since variables cannot be passed along. Writing a program state to a disk file to be used by the next module might be acceptable on a hard disk system, but too slow on a floppy based one.

Next month we'll discuss the features of forking another process that runs concurrently with the parent process. Until then.

/*——— /\/\ark ———*/
(Mark Griffith)

<EOF>

(Example 2)

```
main()
  {
    float a,b,c;
    a=22;
    b=44;
    c=multiply(a,b);
    printf("%d times %d is %d\\n",a,b,c);
  }
  float multiply(a,b)
  float a,b;
  {
    return a*b;
  }
```

The "float" type specifier added to multiply() does two things: It lets main() know that multiply() is returning a "float", and it forces multiply() to actually return a "float".

So far, all is simple. But what happens if we use separate files to compile our code, or if multiply() is a library routine?

As written above, main() will NOT know that multiply() is returning a "float". Since we didn't tell it otherwise it assumes that an "integer" is being returned. Since a "float" is being returned, chaos is assured. The simple solution is to declare our troublesome function in main().

Changing the first line to:

```
float a,b,c,multiply();
```

will work.

From this evolves header files. If you look at any C code you will see that various header files are included in the code via the "#include <xyz.h>" directive. Common include files are "stdio.h", "strings.h", "time.h", "math.h", etc. These files, in addition to other things, declare the return values of various functions. For example, if you list the file "math.h" you will discover that sin(), cos(), and many other functions all return a "double" value. If you are going to do math using these functions you include the header file alerting the compiler of what is really going on.

So what does this have to do with malloc()?

Malloc() is a function which returns a "pointer to char". In most environments (including OS9/6809 and OS9/680xx) the internal size of an integer and a pointer are identical (2 bytes on the 6809, 4 on the 680xx). Because of this, programmers have, historically, been quite cavalier about converting between integers and pointers. But, this practice is dangerous.

What happens when you port your program to an environment where pointers and integers are NOT the same size? Yup—problems.

Neither C compiler package from Microware (6809 or 68K) include a header file for the malloc() function. The manuals advise that malloc() (and associated functions) return a "pointer to char". So, to do things correctly when you use malloc() you should have a statement somewhere in your program which looks something like:

```
extern *char malloc();
```

This works well—so let's save a bit of typing and create out own header file for malloc(), etc. We'll call it malloc.h:

```
char        *malloc(),
            *calloc(),
            *realloc(),
            *ebrk(),
            *ibrk(),
            *sbrk();
```

Including this file in our programs make them nicely "legal". More importantly, they become more portable.

Now, we can have code like this:

```
#include <malloc.h>
foo()
{
   char *p;
   p=malloc(1234);
   ....
}
```

But what happens if we want malloc() to allocate space for something other that "char"? Maybe we need to allocate memory for an array of integers. In this case we need to do something like:

(Example 3)

```
#include <malloc.h>
        woof()
{
   int *i;
   i=malloc(100*sizeof(int));
   ....
}
```

In this case we are allocating memory for an array of 100 integers. However, when we compile the program we get a warning of an "illegal pointer/integer combination". The compiler expects malloc( ) to return a "pointer to char", but here we are assigning that to a "pointer to int". Something's fishy.

One solution is to use a cast in the assignment:

```
i=(int *) malloc(...
```

will keep the compiler happy—and will force the compiler to do any necessary conversions, if necessary (none are needed in this case).

However, if you are using OS9/68000 or a compiler compliant with the ANSI standards there is another way. This "new" C introduces a "generic pointer" which can point at anything. This new pointer is the type "void *". Declaring functions like malloc() to return "void *" instead of "char *" moves you along the path to ANSI C. It also saves a whole bunch of casts in your program. In ANSI C functions like malloc() are declared in the file "stdlib.h". If you are porting UNIX programs, you'll find that the more traditional flavor of C used assumes you have a "malloc.h" file. So why not have a "stdlib.h" which starts off by including "malloc.h"—the best of both worlds.

If we change the declaration our "malloc.h" file from "char *" to "void *" we can avoid the cast statement above. The following will work fine:

ing is chain(). OSK adds another "C" function called chainc(). In this discussion we'll just talk about implementing a chain in "C". Chain for OS-9/6809 is called as:

```
chain(modname, paramsize, paramptr, type, lang, datasize)
char *modname, *paramptr;
int paramsize, type, lang, datasize;
```

Where "modname" is a pointer to the name of the program to run, "paramptr" is a pointer to the parameter string for that program, and "paramsize" is the length of the parameter string. "Type" is the type of program to run, usually set to zero to mean any type. "Language" is usually set to one to indicate a native code (machine language) module.

The different types of modules, their codes and language codes can be found in the file "module.h". Below is an extract of the language types:

```
#define ML_ANY          0
#define ML_OBJECT       1
#define ML_ICODE        2
```

Most of the time, you'll be using ML_OBJECT as both the type and language values. Datasize is usually set to zero to let the new program setup its own data area. Any value given for "datasize" must be in pages (256 byte chunks) and not bytes. This value is then used to calculate additional memory for the new process. This is essentially the same as adding the extra memory pages option to the command line when running a program from the shell.

To chain to another program called "print_report", we can set up ourtest program like so:

```
#include <module.h>

main()
{
    char *prgnam = "print_report";
    char *parms  = "-p p1";

    /* do some processing here */

    /* ready to chain to the print_report
       program */

    chain(prgnam, strlen(parms), parms,
          ML_OBJECT, ML_OBJECT, 0);
}
```

Lets now say that the print_report program is a shell script. Since it is not a machine language program or BASIC09 I-Code, it much be called so as to use a shell to interpret it. To do this, you make the name of the program to run "shell" and add the "print_report" shell script as part of the parameter line like so:

```
char *prgnam = "shell";
char *parms  = "print_report >/p ";
```

This is also a good method to use to run BASIC09 I-Code files. Just use "runb" as the program name and the I-Code file as part of the parameter string. Notice the extra space at the end of the parameter string. This is required for this to work. Apparently, chain or shell (I'm not sure which one) lops off the last character in the parameter string

CHAINING TO NEW PROCESSES UNDER OS9 AND OSK

BY MARK GRIFFITH

One of the nicest and most useful features of the OS-9 operating system, like its UNIX counterpart, is the ability to start another program, or process, from within a currently running program. All computer systems do this to some degree. MS-DOS, for example, runs a program called "command.com" at bootup which puts up the "C:>" prompt and waits for a user to enter a command to run. Command.com then loads the program given and jumps execution to the beginning of that program. When this program is finished, control returns to command.com. RSDOS on the CoCo does something similar except it runs commands that are in ROM which in turn load and execute a program. The end result is the same. The difference between single-tasking operating systems like those running MS-DOS and multi-tasking systems is both programs can run at the same time. The relationship between the two is commonly termed the "parent" and "child" and the procedure for starting another program is called "forking" or "chaining", depending on how you do it.

This month, we'll cover chaining to another process. Forking will be covered in the next issue.

Chaining to another program means to start the new program in place of the one you are running now. This is similar to what the single-tasking operating systems do, with the exception that when the new program is finished, control never returns to the program that called it since it is no longer in memory.

The best reason to use chain, especially within OS-9/6809 systems, is it allows the programmer to start another program on a system with limited memory. It also uses less processing overhead so it is slightly faster then a fork system call. The majority of the time, fork is the preferred method to start a child process, but there time when chain is more suited. One situation would be if the programmer does not want execution to ever return to the program calling chain. This is useful for utilities such as login.

Chain passes on any opened data paths, such as standard out, in, error, and any disk I/O paths. However, using anything other than the standard I/O paths is not for the faint of heart and you'd better be sure of what you are doing. It's best to just open the paths you need within the new process.

Chain first quits the currently running process and then starts the new one in essentially the same memory space. The system call to do this isF$Chain and the "C" bind-

```
#include <malloc.h>
        woof()
{
   int *i;
   char *p;
   i=malloc(100*sizeof(int));
   p=malloc(1234);
   ....
}
```

This is not completely legit if we want to follow the Microware manuals; however, it will not create any unwanted side effect, it makes our programs more compliant with the move to ANSI C, and it saves some typing.

Ultimately, the choice in how to handle declarations (and a whole bunch of other stuff) lies with you, the programmer. "Experts" (which I don't consider myself to be) can only suggest and show alternates. The best way to learn is by doing, making mistakes, and learning from the mistakes.

If you have any comments on this subject, or other columns I've written drop me a note. I can be reached at PO Box 355, Porthill, ID 83853 or PO Box 57, Wynndel, BC, Canada V0B 2N0 or Compuserve 76510,2203.

-Bob van der Poel

<EOF>

REVIEW

DataDex

Review by Leonard Cassady

I've seen many data management programs over the course of time. Few programs live up to the claims stated in the advertising. DATADEX proves to be an exception.

While DATADEX will not fulfill all the database management needs of a small business, it does indeed emulate a Rolodex card file system with such ease of operation, simplicity, and efficiency it is hard to imagine a better implementation.

DATADEX requires a OS9/68000 system with at least 100k ram available plus whatever additional ram is needed for your favorite text editor. A terminal or monitor with at least an 80 x 10 display, and you're ready to run. A hard drive is also recommended for faster media I/O, although it is not necessary as during operation, the files are loaded into memory.

DATADEX comes on a 3.5 micro floppy using the "univ" format. The install file automatically copies the necessary files to the destination device you supply via the command line.

Developed with OS9/68000 v2.4, I immediately expected problems as my system uses the v2.3 operating system. This trepidation proved to be unfounded.

The author wisely supplied a second version of the program with the I/O routines already linked for those using a different revision of the operating system.

The manual is clear, concise, and takes a functional approach, which is a relief from the ones that give you endless examples without much practical information. There is a number to call in case of difficulties encountered, however, the manual should answer any questions.

My only criticism of the manual is the assumption that the user will have practical working knowledge of the termcap file or that other programs have already supplied some of the necessary key definitions and screen control codes. Keep your operating system manual handy just in case.

The setup utility "ddxsetup" and a supplemental utility, "ddxtermtest", notify you of any missing, or conflicting termcap entries. There is also an alternate method to re-define the key sequences in the event your terminal doesn't have function keys or they have non-standard definitions. There is an termcap example for the MM1 and KWindows supplied which can be used as a guide.

Once the termcap file is properly setup, DATADEX is extremely user friendly. The HELP lines along the bottom of the screen inform you of what options are available and the key strokes for those particular functions, (functions vary depending on the screen you are using).

Next, you must setup various environment variables, all of which are outlined in the manual in the technical information section, along with the necessary termcap definitions.

You are given your choice of which text editor to use. I don't recommend using "UMacs" with DATADEX, as UMacs forks a shell upon exit and won't return to DATADEX unless you implicitly kill the UMacs process. The line editor "EDT" works just fine and considering the nature of the DATADEX management program, seems to make an ideal team.

The X_Cmd function, (external command), is touted to be one of the most powerful features of DATADEX. When the command is forked, DATADEX will pass up to ten user defined parameters. DATADEX handles the piping and data format sent to the X_Cmd for you. Two C source code examples are included on the disk, and may be used as a guide for writing your own X_Cmds. Again, this shows the authors' foresight in that you may compile the X_Cmd using the your operating system version level.

The utility, "ddx2txt", allows you to extract or "stack" cards for backup to a text format file for spell checking. The utility, "ddxpr", allows you to send the card data to a printer for hardcopy. Both of these utilities may be invoked from within the X_Cmd function. Also included is a utility, "mak4ddx", that will translate text files into the DATADEX format, (with minor preparation to the text file). The ability to fork a shell comes in handy at this point, although most of the functions desired may be accessed through X_Cmd. Spell checking using the public domain port of "ISpell" was flawless. Any spell checker that uses an ASCII format should work equally as well.

One particular function that may be very useful is the Global Search. DATADEX searched through one hundred cards for a target string within two seconds. A very impressive performance.

DATADEX showed the same efficient performance running under GWindows, once the termcap file was properly setup. Unfortunately, KWindows was not available for this review, although I have little doubt that any difference could be noted.

The Multi-user version will be of interest to those running BBS's, and at the list price for the single-user version, DATADEX is well worth the price, a true bargain. I highly recommend it over the more expensive datamanagers for the power, user -friendliness, versatility and cost.

Leonard Cassady (Maudib)

<EOF>

SUBMIT!

Submissions to the **OS9 Underground** are always welcome from you.

Submissions may be sent to the Magazine by Email or on Disk by regular Mail to our address (See Table of Contents page).

Please send your Email submissions to:
zog!sysop@abode.ttank.com
zog@delphi.com
or "ZOG" on Delphi

string array assigned to it.

This next example of a BASIC program dimensions an array that will hold up to 51 strings. A loop is entered and user input is received from the keyboard on each pass.

```
10 DIM A$(50)
20 FOR X = 0 TO 50
30 INPUT A$(X)
40 NEXT X
```

A possible C version is:

```
main()
{
    int x;
    char *a[51], *malloc();

    for(x = 0; x <= 50; ++x)
        gets(a[x] = malloc(256));
}
```

However, this translation is wasteful. The input string probably won't be 255 characters in length. To eliminate the wasted memory space, a better version would look like this:

```
main()
{
    int x;
    char *a[51], b[256];

    for(x = 0; x <= 50; ++x)
    {
        gets(b);
        a[x] = malloc(strlen(b));
        strcpy(a, b);
    }
}
```

Of course this example assumes you won't be performing any other operations on the pointer, such as copying another string to the end of the input string. The memory allocated is big enough only for the input string.

As stated earlier, the "calloc( )" function does about the same thing as the "malloc( )" function, except that it sets the bytes in the reserved memory block to zero, or initializes the memory block. This aspect is very handy for resetting the variables in a structure or union.

The "realloc( )" function changes the size of the memory block previously allocated by either "malloc( )" or "calloc( )". If the new size is smaller than the old size, the unused portion of the memory block is discarded. If the new size is larger than the old size, all of the old contents are preserved and new memory space is tacked onto the end. If the new memory space cannot be allocated, no new space is allocated and "realloc()" returns a
null pointer.

After the memory block is allocated, used, and no longer needed, it is always desirable to return the memory space back to the system. The general rule should be, if you don't need it any more, get it out of the way. If this isn't done, we could possibly run out of memory or overrun the stack. The "free()" function releases the allocated memory and sets the pointer to a null.

Pointers operations in C make up a large part of the it's flexibility. The unusual nature of pointers can cause a BASIC programmer much confusion at first. Once you begin to understand pointers, you will see the power and flexibility they provide and miss them if you find yourself in an environment that doesn't support them.

Maudib (Leonard Cassady)

C Software Engineering

by Leonard Cassady

MEMORY ALLOCATION

Pointers can be made to point to any area of memory. Usually, the locations pointed at have already been written to or is already in use by the program or the operating system. When the need for an area of memory set aside specifically for storage of a large amount of data and pointer access is desired, memory allocation comes into use. This can be done by declaring a large char array, but array space has limits. Once used up, the space must be enlarged, (if possible), or another method is needed.

The standard C function set includes the memory allocation functions, "malloc( )", "calloc( )", "realloc( )", and "free( )", which allow areas of memory to be set aside and accessed by a pointer, resized and returned to the operating system for other uses. The functions attempt to locate an area of memory not in current use. When an area or block is found, the function allocates or marks the desired number of sequential bytes as being used and returns a pointer to the first byte in the block.

While there may be other memory allocation functions available for your compiler, they are not part of the standard C language set, and may not be portable to other systems.

The "malloc( )" function allocates a block of memory and assigns a pointer to the first byte. Nothing is done to contents of the block, so whatever garbage was there before the allocation still resides there. The "calloc( )" function does about the same thing except that it clears, or sets to zero, each byte in the block.

The following C program demonstrates the use of "malloc( )".

```
main()
{
   char *a, *malloc();

   a = malloc(100);
   strcpy(a, "OS9 Underground");
}
```

The example sets aside a block of 100 bytes. the char pointer "*a" is assigned to the the first byte in the block using the memory location return by "malloc()". The next line uses "strcpy()" to copy the string assignment to the memory location pointed to by pointer "*a".

While it is unwise to copy anything to pointers because it could result in an overwrite of memory already in use, pointer "*a" now has 100 bytes reserved for it's usage. Plenty of memory storage for the

BSpeed

by Paul Pollock

OK folks, some of you have asked me to put some programming where my mouth has been <grin>. So for your programming enjoyment, what follows is the docfile* and program-source for BSpeed r1.01. This program has been published on StG-NET before, but for those of you who didn't pick it up, here's everything you need to know about it.

I don't program in machine language ('C', Pascal-object, Assembler; etc.), so I've made it my business to find ways of doing interesting things with Basic09, without sacrifice to speed or efficiency. You'll find the utilities I write work every-bit as easily and fast as machine language programming. And because they're Basic09, you can understand them, and modify them to your heart's content.

Normally my source files are only lightly sprinkled with remarks. But for this article, I've gone back into the program to thoroughly comment it. This is done so that you may understand every algorithm inside the program. I hope you can not only use the program, but that it helps you Basic09 newcomers to do your own programming better Just for your edification, BSpeed r1.01, is the fastest drive throughput benchmark in OS9 Level-2. Period ☺!

NOTE: This program is intended for OS9 Level-1 and Level 1-2. Those of you who have OS9/68000 system, may find this program useful after patching it to operate on longer buffer, media lengths and altering the LSN0 data appropriate to your system. It has already been tested on a FHL-TC70, after some rework, and it is known to work. OSK users may find the program more efficient by extending the buffer from 25k to 50-100k. Also extend the media read from 1 meg to 20-40 meg, to take into account the much faster drive throughput. Especially important on hard disk drives. Floppy drives will require few of these changes as the drives are mechanically the same throughput in OSK as they are in OS9 Level-1 and OS9 Level-2.

[***Note: Because of space considerations, the Entire DOC File for BSpeed will be made available on the Underground-on-Disk or may be downloaded from our BBS directly. Our BBS Number is (818) 769-1938**]

PROGRAM DESCRIPTION

'BSpeed' is a Basic09 program which can calculate the actual throughput of ANY drive media.

'BSpeed' handles all the internal housekeeping, including the obtaining of the requested drives' storage capacity from LSN0, the storing of start-time and end-time, in a manner which has no interference with the actual drive read. Also handled is the procedure of starting the media read, by presetting the heads at track-0/sector-0, then proceeding to read all available media (up to and including 1000 KBytes); in 25-KBYTE size blocks.

'BSpeed' actually reads the drive faster than the 'Megaread' program, by Bruce Isted, since it reads in larger blocks of data than 'Megaread', and can take advantage of fewer program and system interruptions to data flow.

PROGRAM OPERATION

'BSpeed' is used like any standard utility. But its syntax variesslightly, depending on whether the user has the standard SHELL or SHELL+.

For users of standard shell (or in Basic09);

```
OS9:BSpeed("/devicename") <ENTER>
      (ie, /d0,/h0,/r0; etc)
```

For users of SHELL+ (any type);

```
OS9:BSpeed /devicename <ENTER>
```

'BSpeed' then titles the screen, gets the media capacity from the media under test. Correct storage capacity, will be calculated, even when drives have media installed, which have been formatted to standards which differ from the potential of the hardware. It then sets the drive-under-test to '0-0', sets the start-time, and begins reading the media under test. Once the media is read, it then sets the end-time, and calculates the total throughput per second.

```
PROCEDURE BSpeed
(* BSpeed r1.01
(* Copyright (c) May 1991, Paul Pollock,
(* All Rights Reserved
(*-----------------------------------------
(* BSpeed properly reads any size disk,
(* in a serial fashion, without damaging
(* the disk being tested; at the maximum
(* speed of the drive under test.
(*
(* BSpeed is a smart interface that uses
(* this test in a fashion which allows the
(* testing of any size drive and media.
(*
(* BSpeed will calculate media size, time
(* to read up to 1000 KBYTES, and then
(* calculate total thruput in KBYTES per
(* second.
(*
BASE 0
 ON ERROR GOTO 1000
PARAM drvin:STRING
(*---------------------------------------
DIM buffer(25600):BYTE
(* I Note two buffers. One for buffer

DIM LSN0(256):BYTE
(* I loads to read maxK. Other to
(* gather LSN0 sector data.
(*
DIM a$,b$,drvptr:STRING
DIM bufferK,maxK,entry,tot,a,b,c,d:REAL
DIM path:BYTE
(* Following is explanation of all above
(* variables a$,b$ is date$'s.
(* drvptr is tranfer variable for 'drvin'.
```

CANAWARE

Ingenuity from the "Great White North"

ENC9 IS A SIMPLY TWO ENTRY DATABASE WHICH YOU CAN USE TO STORE PHONE NUMBERS & ADDRESSES, RECIPES, DEFINITIONS OR ANYTHING ELSE YOU'VE SEEN AN ENCYCLOPEDIA USED FOR. STORE OVER 250,000 RECORDS OF 8000 BYTES EACH.

FEATURES INCLUDE:

- BUILT-IN TEXT EDITOR
- IMPORT/EXPORT TEXT FROM/TO EXTERNAL TEXT FILES
- PRINTOUT INDIVIDUAL ENTRIES OR THE FULL DATABASE
- ADD, DELETE, RETRIEVE AND SEARCH RECORDS

REQUIRES OS 8 LEVEL II/COCO3 W/ 512K 80 COLUMN MONITOR, RUNB, SYSCALL, DISPLAY AND TMODE.

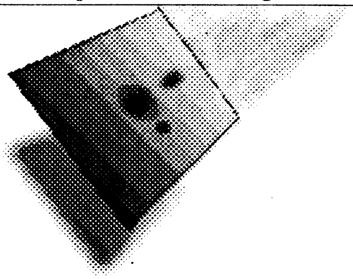
Only **$24.95** (U.S.) + $2.00 S&H

$29.95 (CDN) + $2.00 S&H + 7% GST
ONT. RES ADD 8% PST
FOREIGN ORDERS $4.00 S&H

SEND CHEQUES OR MONEY ODERS TO:
CANAWARE
1378 CREDIT WOODLANDS COURT
MISSISSAUGA, ONTARIO, L5C 3J5 CANADA
(416) 279-4395

```
a=VAL(MID$(a$,10,2))*3600+
        VAL(MID$(a$,13,2))*60+
        VAL(RIGHT$(a$,2))

 b=VAL(MID$(b$,10,2))*3600+
        VAL(MID$(b$,13,2))*60+
        VAL(RIGHT$(b$,2))
```



- Paul Pollock

```
(* Above two lines convert date$'s to
(* numeric data, for later calcs.
(* The extremely large MID$ arguments
(* are done to allow proper program
(* usage, no matter what time of day!

IF b<a THEN
(* This repairs conditions if the times
b=b+86400.
(* I gathered, between one day and
(* another.
ENDIF

c=b-a
(* This is it! Elapsed time calc!
PRINT
PRINT "Total time in seconds= ";
PRINT USING "r11.2>"; c
PRINT "Thruput in KBYTES/Sec= ";
PRINT USING "r11.2>"; entry/c
END

(* What follows is the program error-trap.  Used mostly to print help message
(* if drivename not provided as param;  or is attempted incorrectly.
1000 er=ERR
IF er=211 THEN 10
(* I Impossible! Left in as SAFETY.
PRINT
PRINT "USAGE:"
PRINT "     OS9:BSpeed(""devicename"")"
PRINT "devicename=/d0, /h0, /dd, /r0;  etc."
PRINT
ON ERROR
ERROR er \ (* Hint: Forces fork to F$Perr Systemcall.
END
```

<EOF>

```
(* bufferK is size of 'buffer', in kbytes.
(* maxK is size of drive-read, in kbytes.
(* entry and tot are transient reused
(* variables for drive-calcs and buffer
(* reads.  a,b,c,d are transient variables,
(* d is a loop variable, a,b,c are mostly
(* time-calcs, path is used to open the
(* drive under test.

a$="" \b$="" \drvptr=""
a=0 \b=0 \c=0
d=0 \tot=0 \entry=0

(* Soft-items. Change these to alter
(* bufferK=25.  I global factors.
(* Note: change 'buffer' maxK=1000.
(* AND 'bufferK' as a pair!
(* 'maxK' is simple stuff.
(*
(* PRINT
(* PRINT "BSpeed r1.01"
(* PRINT "Copyright (c) May 1991,
            Paul Pollock"
PRINT "————————————————
————"
(*
drvptr=drvin  \ (* Used to trip Error-Trap
(*
PRINT "Device name="; drvptr
(*
OPEN #path,drvptr+"@":READ
(* This stuff used to open
SEEK #path,0
(* drive 'raw-mode', and get
GET #path,LSN0 / (* LSN0 on drive.
SEEK #path,0 / (*  Note:Drive stays open!
entry=FLOAT(LSN0(0))*65536.+
      FLOAT(LSN0(1))*256+LSN0(2)
(* Above line is the initial calc, determines
(* number of   sectors on drive. Used later
(* for 'tot'; etc.
(*
PRINT
PRINT "Total sectors on media= "; entry
(* Sector=256bytes. Sector*4=Kbytes.
entry=entry/4

(* Sectors on drive/4=number of kbytes
(* on drive.

PRINT "     KBYTES on media= "; entry
IF entry>maxK THEN
(* Ensures program tests up to but not
(* exceeding 1meg(maxK).
entry=maxK  \ (*'maxK'
GOTO 5 \ (* must be multiples of 'bufferK'.
ENDIF

tot=entry \ (* All else failed.
entry=FIX(entry/bufferK)*bufferK
(* This rounds to nearest possible
(* 'maxK'/'bufferK'.
IF entry>tot THEN
entry=entry-bufferK
ENDIF
tot=entry
PRINT "     KBYTES under test= "; tot
tot=tot/bufferK
(*  Determines # of buffer-loads.
a$=DATE$
(* Ensures that timing of drive-read begins
(* at a rollover of second!
REPEAT
UNTIL DATE$<>a$ I

(* First stored time! Used to find
(* elapsed time used after second
(* stored time.
a$=DATE$
(*  Actual drive-read!  This is the main
(* reason for whole program!
FOR d=1 TO tot
GET #path,buffer
NEXT d          I
CLOSE #path        I

(* Enter your clock reset routine here!
10  b$=DATE$
(* Second stored time. Determines Elapsed
(* Time!
```

Also available from Gale Force

From Clearbrook Software Group

MSF	MSDOS file manager for OS9 - makes transferring files between MSDOS and OS9 a snap. Requires OS9 LII and SDISK3.	$35.00
IMS	Complete and powerful database for OS9 LI and LII Previously retailed for $169.00!!	$65.00
Erina	Program debugger. Very powerful and loaded with features. For OS9 Level I and II.	$45.00
Serina	System module debugger. Find those hard to track bugs in device drivers and other system modules. For OS9 LEvel I and II.	$65.00

From D.P. Johnson

LI Utility Pak	A very extensive collection of useful utilities. You will wonder how you ever managed without them.	$49.95
LII Utility Pak	An extension of the L1 Utility Pak More useful utilities.	$39.95
BootFix	Boot from a double sided disk under OS9 Level I. Requires SDISK.	$9.95
SDisk	A replacement floppy disk driver for OS9 Level I. Allows the use of double sided drives.	$29.95
SDisk3	A replacement floppy disk driver for OS9 Level II.. SDisk3 has provisions for reading non-OS9 disks.	$35.95
Forth09	Complete Forth compiler for OS9 Level I and LEvel II.	$150.00

Experience GALE FORCE speed!

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

Send cheque or money order to :
Gale Force Enterprises
P.O. Box 631, Surrey, BC,
Canada, V3T 5L9

(604) 589-1660

8 AM - 5PM PST (voice)
5 PM - 8 AM PST (support BBS)

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

JWT Enterprises

Optimize Utility Set 1:

➡ Optimize your floppies and hard drives quickly and easily! ➡ Includes utility to check file and directory fragmentation. ➡ Works alone or with Burke & Burke *repack* utility. ➡ One stop optimization for any Level 2 OS-9 system.

$29.95; Foreign Postage, add $3.00

Optimize Utility Set 2:

➡ Check and correct any disk's file and directory structures without any technical mumbo-jumbo.
➡ Run periodically to maintain the integrity of your disks as well as the reliability of your data.
➡ Especially useful before optimizing your disks.

$19.95; Foreign Postage, add $3.00

Optimize Utility Set Pac:

➡ Get both packages together and save!

$39.95; Foreign Postage, add $4.00

NINE TIMES

Nine-Times: The bi-monthly disk magazine for OS-9 Level 2.

In each issue:
- Helpful and useful programs
- C and Basic09 programming examples
- Hints, Help columns, and informative articles
- All graphic/joystick interface
- Can be used with a hard disk or ram disk

One Year Subscription, $34.95;
Canadian Orders, add $1.00; Foreign Orders, add $8.00

Back-Issues: From May 1989. Write for back issue contents.
$7.00 each; Foreign Orders, add $2.00 each

Magazine Source: Full Basic09 code and documentation for the presentation shell used with Nine-Times.
$25.95; Foreign Orders, add $5.00

Technical Assistance & Inquiries:
(216) 758-7694

JWT Enterprises
5755 Lockwood Blvd.
Youngstown, OH 44512

RAINBOW CERTIFICATION SEAL

Foreign postage excludes U.S. Territories and Canada. These products for OS-9 Level 2 on the CoCo 3. Sorry, no C.O.D.'s or credit cards; Foreign & Canadian orders, *please* use U.S. money orders. U.S. checks, allow 4 weeks for receipt of order. Ohio residents, please add 6% sales tax.