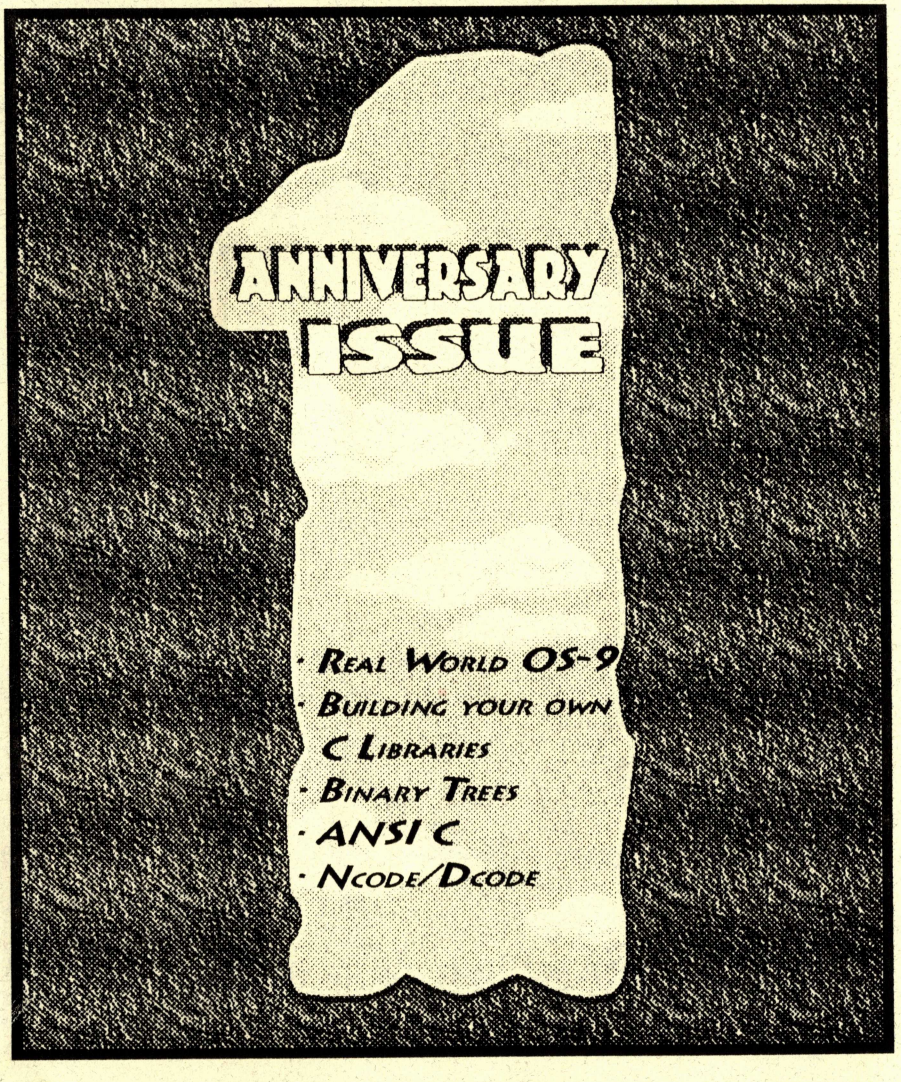
Volume Two, Issue One

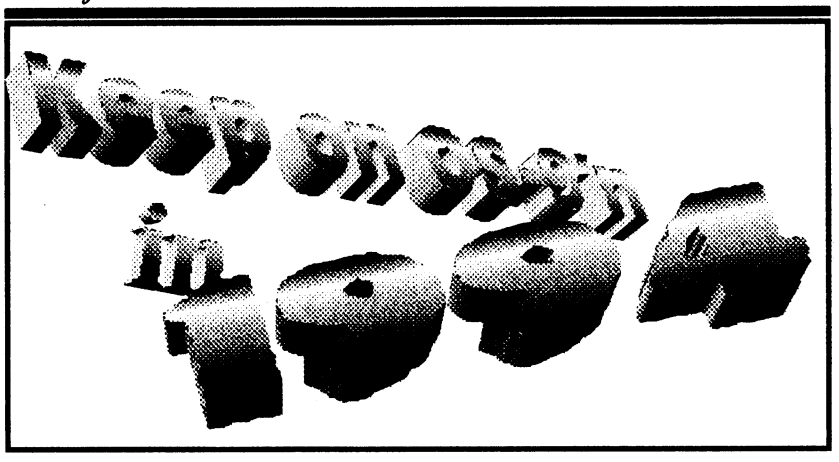
$3.00

The "International" OS9 Underground®

A Fat Cat® Publication

MAGAZINE DEDICATED TO OS-9/OSK USERS EVERYWHERE!





Advertisers Index

Let these fine vendors know you saw it in The Underground!

The "International" OS9 Underground® Magazine

Dedicated to OS-9/OSK Users Everywhere

Volume Two, Issue One

CONTENTS

The Underground Staff

Editor/Publisher:
Alan Sheltra

Assistant Editors:
Jim Vestal
Steve Secord

Technical Editor:
Leonard Cassady

Contributing Editors:
Wayne Campbell
Allen Huffman
Andy Jacobson
Scott McGee
Colin McKay
Eric Levinson
Boisy Pitre
Bob van der Poel

Typesetting/Layout/Artwork:
AniMajik
Productions

Mail Room — Letters to the Editor

Wants to Score a ScoreMod

Dear Editor,

In Issue #10 there was a tic tac toe program. You did not include scoremod.c program listing in that issue and said it would be in the next issue. I haven't seen it yet, will it be in a future issue?

-Keith Bauer, Memominee, MI
(received from Delphi via AOL internet gateway)

Keith,

Lack of space prevented the the listing from appearing in issue #11 or #12 as was planned. The listing "ScoreMod.C", however, does appear in this issue (Page 36).

Listings may always be requested via internet (I have forwarded about 6 requests in the last month, Keith was one of them!). See page 15 for information on where to send requests.

-Editor

Glaring Error

I was just going over Cassady's article in the Dec. issue and have found a glaring error which should be commented on in the next issue! On page 10 in the discussion on strcat() it states that memory is checked and automatically allocated for the operations. It ain't so...and if one relies on that you'll get wonderful program crashes!

-Bob van der Poel, Port Hill, ID
(Received from CIS via AOL intern gateway)

Comments, Leonard?

-Editor

Trashed File

I was just about finished with my article on how I simulate virtual memory and break the 64K barrier under OS9 Level II, but somehow the file was trashed on n coco. So I'm starting from scratch. What a bummer, I was 90% finished. It's really frustrating to get so close to being finished and then lose it. Basicly, I just wanted to let you know I hadn't forgotten. BTW: Your mag is really starting to look professional. Something that might be helpful to the community would be a listing of all ftp and internet listserv sites that support OS9. Currently, I coco@pucc.princeton.edu, and a mailserv at OS9archive@cabrales.cs.wisc.ed Also, I noticed you have an account on America Online. I'm thinking of joining either GEnie or America Online. Do you know which has the better OS9 group.

-Chris Strickland,
New Horizons Software
(407) 264-9328

Chris,

Sorry to hear about all that lost wc Chris... yowch! I know how frustrating that can be.

I like the idea of listing ftp sites for OS-9. Look for something in a future issue.

Between GEnie and AOL, I'd go wi GEnie for the better OS-9 Group. I use AOL mainly for email.

-Editor


```
for ( k = 0; k < 3; k++ )
  if ( a[k][k] == 'X' )
   xcount++;
  else if ( a[k][k] == 'O' )
   ocount++;

 return( evaluate(xcount, ocount));
}

int diag2score( a )
char a[][3];
{
 int k;
 int xcount = 0;
 int ocount = 0;

 for ( k = 0; k < 3; k++ )
  if ( a[k][2-k] == 'X' )
   xcount++;
  else if ( a[k][2-k] == 'O' )
   ocount++;

 return( evaluate(xcount, ocount));
}
```

(EOF)

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

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

The Glenside Color Computer Club of Illinois presents...

The 3rd Annual 'Last' CoCoFest

May 21st & 22nd, 1994

Holiday Inn Elgin (A Holidrome Indoor Recreation Center)
345 W. River Road (A city block from I-90 & IL-31S)
Elgin, Illinois *(Same great location as last year!)*

Overnight room rate $52.00 (plus 10% tax)
Call 1(708) 695-5000 for reservations
Be sure to ask for the **"CoCoFest"** *rate!*

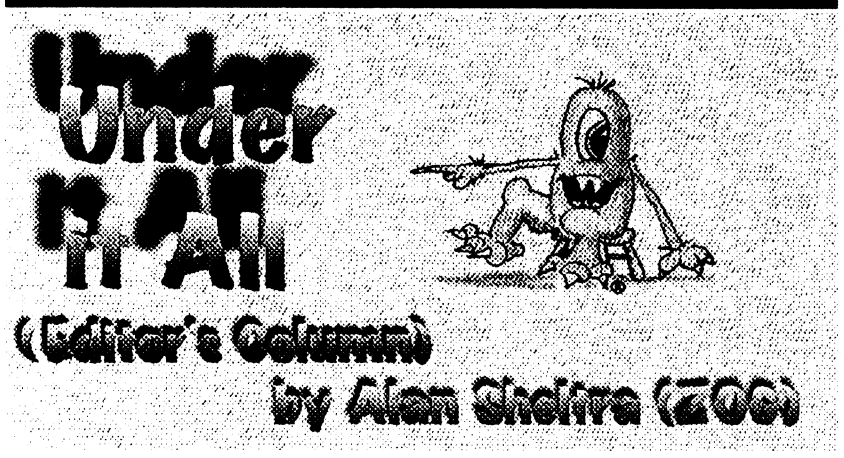
Admission is $15.00 at the door (2-day pass only)
Advance ticket sales $10.00 + SASE (or $10. + 50¢ P&H)

George Schneeweis, Treasurer,
Glenside Color Computer Club
RR#2, Box 67, Forrest, IL 61741-9629

Tony Podraza, Fest Chairman
(708) 428-3576 (Voice)
(708) 428-0436 BBS
Delphi: **TonyPodraza**

Carl Boll, Vice President GCCCI
(312) 735-6087 (Voice)
(312) 735-3355 (BBS)
Delphi: **CBJ**

Brian Schubring, President GCCCI
(708) 529-3539 (voice)
Delphi: **THESCHU**



New Volume, New Year!

Happy New Year! I sincerely hope 1994 brings you the best.

The Underground starts the new year with our ***Anniversary Issue*** and with 12 issues under our belt (13 including this one) we're ready to continue to provide the *best* articles, tutorials, news and views of OS-9 for another year.

We intend to continue making improvements and refining this publication even more as time goes on... look for more changes in the near future.

First Class

I hear one publication is going to 3rd class mail. As a cost conscious editor/publisher, I looked into the idea... and totally rejected it! The Underground will ALWAYS be sent First Class Mail.

The U.S. Postal Service does treat 1st class mail much better than it does 3rd class mail, and the cost difference isn't worth the headaches the cheaper way could cause. Besides, first class mags, gotta go first class ☺!

TheFatCat List

The Underground has it's own email list (sorta like the CoCoList at Princeton) and everyone is welcome to participate (it's free, see page 15 for details). V occasionally have contests on-line. In fact we just had 3 winners from our New Years Contest. Congratulations to First Prize winner Timothy A. Johns!

The Return of the MM/1

David Graham, president **BlackHawk Enterprises**, has announced a special deal to buy an MM/1. H asking a deposit of only $10.00 to the first 20 people to order one of his 3 available MM/1 systems. In return for being one of the first 20, he is offering a special discount. Best of luck to David a BlackHawk. (See BlackHawk's ad on page 8.)

The Underground has decided to take advantage of this deal and has send in an advanced order. The MM/1 will make a nice addition (along with o present TC70), to our programmi arsensal and review arsenal.

Contest

Next month, we'll kick off the year with another contest. Until then, *Keep on doin' it in real-time!*

*-ZOG (Alan Sheltra)

Tic Tac Toe

(Continued from October 1993 Issue)

by Jim Vestal

```
/*
This file contains the function score and other
functions that it calls to evaluate the score for
a tic-tac-toe square.
To use it, write prog5.c using the prototype
extern int score( char[][3], int, int );
Then compile prog5.c and scoremod.c as
follows :
cc -Aa -c scoremod.c  <- creates scoremod.o
cc -Aa -c prog5.c     <- creates prog5.o
cc -o prog5  prog5.o scoremod.o <- links the
two .o files and creates the executable prog5
To execute your program, give the command
prog5
*/
int score( a, r, c )
char a[][3];
int r;
int c;
{
  int total = 0;
  total += rowscore( a, r );
  total += colscore( a, c );
  if ( r == c )
    total += diag1score( a );
  if ( r + c == 2 )
    total += diag2score( a );
  return( total );
}

int evaluate(x,oh )
int x;
int oh;
{
  if ( oh == 2 )
    return(50);
  if ( x == 2 )
    return(25);
  if ( ( oh == 1 ) && ( x == 0 ) )
   return(10);
  if ( ( x == 1 ) && ( oh == 0 ) )
   return(8);
  if ( ( x == 0 ) && ( oh == 0 ) )
   return(4);
  else
   return(0);
}

int rowscore( a, r )
char a[][3];
int r;
{
 int k;
 int xcount = 0;
 int ocount = 0;

 for ( k = 0; k < 3; k++ )
  if ( a[r][k] == 'X' )
   xcount++;
  else if ( a[r][k] == 'O' )
   ocount++;

 return( evaluate(xcount, ocount));
}

int colscore( a, c )
char a[][3];
int c;
{
 int k;
 int xcount = 0;
 int ocount = 0;

 for ( k = 0; k < 3; k++ )
  if ( a[k][c] == 'X' )
   xcount++;
  else if ( a[k][c] == 'O' )
   ocount++;

 return( evaluate(xcount, ocount));
}

int diag1score(a)
char a[][3];

{
 int k;
 int xcount = 0;
 int ocount = 0;
```

Continued Page 37

EXPRESSIONS
Structure assignment

ANSI C allows the initialization of union variables. If the first component of a union is a structure, the entire structure is is initialized.

The initializer for a static, external, automatic, or register union variable must be a brace enclosed constant expression.

Passing structures as arguments

There are methods of structure passing, by value and by reference. The ANSI C Standard requires the compiler to support passing structures by value.

It is worth noting that there are only two circumstances where passing a structure by value is useful since the entire structure must copied.

- The structure is very small.
- To guarantee the called function does not change the structure being passed.

STORAGE CLASS
Function prototypes

The two main benefits of function prototyping are:

- compiler type-checking of actual function arguments at compile-time.
- automatic argument conversion is turned off.

Prototyping usually reduces the number of conversions needed when passing arguments to functions by allowing specified data types to be passed.

INITIALIZATION
'struct' and 'union' name spaces

The ANSI C Standard requires the compiler to create a separate 'naming space' within each structure and union, so that two or more structures, or unions may have components with the same name.

This feature is a late extension to the K&R Standard and may not be available on older compilers. Some older compilers do allow 'same naming' of components, but they must be in the same location and of the same data type between the structures or unions.

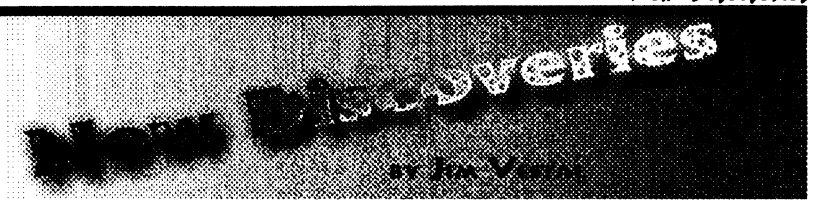
This is the third installment of several in a series on the additions and differences introduced by the ANSI C Standard.

Any comments, suggestions are welcomed and should be sent to:
Software Engineering Assoc.
Standard ANSI C Library
6530 Independence Ave Ste #1(
Canoga Park, CA 91303
or emailed:
"zog!leonard@abode.ttank.com" (or see page 15 for other mail and email addresses)

-Leonard Cassady

(LOF)

Turn to page 37



New Discoveries contains a brief overview of new products, hardware, software (both commercial and shareware/public domain). If you are a vendor or a software author and would like to submit a list of new products please send a general description to Jim Vestal c/o this magazine or via internet email: [os9under@aol.com, discover@narnia.citrus.sac.ca.us or zogster@delphi.com]

Commercial Products:

DDF from ARK Systems USA

A Unix-like device descriptor for OS9/68000 V2.4. (Look for a review of this new product in our next issue). For more information, please contact ARK Systems USA (408) 244-5358. Email: arkusa@aol.com

Pipeline Online

Microware Systems has released a text-only version of their quarterly magazine Pipelines to the OS-9 Comunity. Beginning with Vol. 8 No. 1, Pipelines in archived format can be found on Compuserve and the StG Network. Soon it will be available on Delphi and other major bulletin boards. Thumbs up to Microware!

OS-9 Version 3.0 from Microware

Microware announced it's release of OS-9 Version 3.0, which is available in two kernals, Standard OS-9 and Atomic OS-9. The Standard OS-9 Kernal can be used for both resident developemenet systems and target run-time systems. Atomic OS-9, a 100% compatible subset of Standard OS-9 is strictly for target run-time embedded systems. Both offer higher performance and less system overhead than previous versions of OS-9. Details of OS-9 Version 3.0 is featured in Vol. 8 No. 1 of Pipelines. Contact Microware at 1-800-475-9000 for your free copy of Pipelines.

The Return of the MM/1 - BlackHawk Enterprises

David Graham of BlackHawk Enterprises has just announced a Special Deal for the first 20 people to put down a deposit for a new MM/1 System. A $10.00 deposit will be used to secure a loan to release 50 ready and waiting MM/1's from the factory. Three systems are available, The Developers System, The Professional System and the Extended Kit. Call or write for more information: (405) 234-2347 · P.O. Box 10552, Enid, OK 73706-0552. Email: NIMITZ@Delphi.com

Online Services:

DELPHI and TYMNET

Delphi announced that you can now start calling Delphi via Tymnet at 6:00 pm instead of 7:00 pm with no surcharge. This is good news for people on the West Coast who want to conferance with those on the East Coast. Tymnet evening hours are now from 6:00 pm-6:00 am in your local time zone.

Shareware and Public Domain:

UUTOOLS uploaded by Boisy Pitre (Delphi Name: BOISY)

UUE and UUD are a set of enhanced UUENCODE/UUDECODE tools which can dynamically size output files into multiple chunks and automatically reassemble and decode them into object files. Executables for both OSk and OS-9/6809, and source code/documentation.

Continues

New Discoveries

Shareware/PD Software (cont)

VEFSHOW for the **MM/1** uploaded by **JOELHEGBERG:**
VEF picture file viewer for K-windows. Finally released over a year after Joel wrote it.

JPEG FAQ and source uploaded by **WRHAMBLEN:**
Freqently asked questions and sources to convert **JPEG** image files to and from a variety of other image file formats.

RCIS BBS V.2.3 uploaded by **SROTTINGER:**
Demo version of **RCIS BBS** for **OS-9/68000** is now available on Delphi. Supports external transfer protocols, full featured real-time conferancing, networking, etc.

Fests/Seminars

The Third Annual "Last" Chicago CoCo Fest
May 21st and 22nd, 1994 at the Holiday Inn, Elgin, Il. (same location as last year). Overnight room rate is $52.00 (plus tax). For reservations call (708) 695-5000.
Advance Ticket Sales are $10.00+SASE. Call (708) 529-3539 for mor information. Be Sure to mention COCOFEST when you call for Reservations!

Pacific Northwest CoCoFest IV
The PNW Fest is scheduled for June 24th and 25th, 1994 in Port Orchard, WA. Contact Donald Zimmerman @ (206) 871-6335 or write to 3046 Banner Road S.E., Port Orchard, WA 98366 for more information.

If you'd like to see your OS-9/OSK Product listed here, (Commercial or Shareware), call (818) 761-4135 or Send EMail to: OS9Under@AOL.com or U.S. Mail to 4650 Cahuenga Blvd., Ste #7, Toluca Lake, CA 91602

should flagged as a compile-time error.

The second use of the 'void' data type, while commonly seen in coding, is more subtle.

Prior to the ANSI C Standard, it was the programmers responsibility to cast the returned pointer to a pointer to the correct data type. ANSI C implements the 'void' pointer as a 'generic pointer', and is automatically cast to the correct returned pointer type when assigned a value.

Generic pointers add flexibility when using another ANSI C feature called prototyping. When a function has a formal parameter, the actual argument must be of the same type cast. By using the 'void' pointer type, the formal may be of any type cast.

EG:

```
char *strcpy(char *s1, const char *s2);  void *strcpy(void *s1, const void *s2);
```

The only consideration when using the 'void' data type in this manner is function variable storage allocation. In the above example, the function 'strcpy()' expects a 'char' data type and variable storage limits may be overrun if the value is larger than a 'char' data type.

'enumeration' data types

The enumerated data type is a recent addition to the K&R Standard and other Tradition C implementations. Similar concepts to this data type occur in other languages, such as Pascal and Ada.

Enumerated types are implemented by associating integer values with the enumerated constants. The first enumerated constant receives the value of zero, (0), if no explicit value is defined. Subsequent enumerated constants without explicit value association, receive an integer value one greater than the previous constant.

The ANSI C Standard specifies that enumerated types must be of an 'int' type cast. This strict type checking is not always present in pre-ANSI C compilers and could introduce run-time errors into the program.

STATEMENTS

'switch' expression control

The K&R Standard requires the switch statement expression to be of an 'int' data type.

The ANSI C Standard requires the expression only to be an integral expression and allows data types of 'char', 'short', 'int', or 'long' and the number of case-label expressions supported must be a maximum of 257.

Example:

```
int const *pointer;
```

This defines a pointer to a 'const' data object and ensures the object cannot be modified. However, the pointer address may be reassigned to a different 'const' data object address. Conversely, passing a 'const' pointer to object argument:

```
int *const pointer;
```

ensures the object 'pointed to' cannot be modified and must point to the same 'const' object for its entire duration.

'volatile' qualifier

The 'volatile' qualifier informs the compiler that the data object may be modified in ways the compiler may not recognize and turns off optimization. This ensures the compiler translation of the source code is exactly as it appears in the file and not replaced by the compilers' idea of a more efficient algorithm.

INTEGER DATA TYPES

ANSI C compiler implementations are required to document the ranges of integers in the include header file, 'limits.h'. The compiler is free to substitute its own values, but must support the minimum values and must have the same sign extension as specified in the standard.

These restrictions support systems using the three most common types of binary integer encoding techniques, (internal integer representation).

- Two's Complement Notation.
- One's Complement Notation.
- Sign Magnitude Notation.

'signed' data types

ANSI C implemented the 'signed' data type as part of the standard, having a value range of (-32,767 to +32,767). This allows variables to hold either positive or negative values. Most compilers assign variables as 'signed' data types by default, so use of the 'signed' keyword is usually for informational purposes since the three common notations represent integers identically.

The one exception is the 'char' data type. Your compiler will most likely use a 'signed' char data type by default, however, this determination is left to the compiler developer and might produce unusual results or side effects when using the type 'char', or bit fields in structure Consult your compiler documentation for the default data type.

'unsigned' data types

Each 'signed' integer data type has a corresponding 'unsigned' data type. The 'unsigned' data type is restricted to non-negative values and have double the positive range of the corresponding 'signed' data type, (0 to 65,635). The stora requirements are the same for 'signed' and 'unsigned' integers, however the integer encoding techniques between the two data types may be different. Straight binary notation is used to represent 'unsigned' data types, regardless of tl 'signed' notation technique.

'void' data type

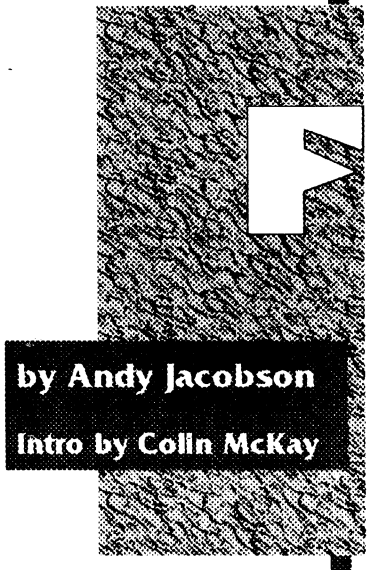
The 'void' data type was not originally part of the C language. The semantics were vague and varied across the many implementations. The ANSI C Standard defines two qualities that must be supported, although the compiler developers are free to include others.

The first use of the 'void' data type, and the most commonly used, is to indicate that a function does not return value, or returns a NULL pointer.

```
EG: extern void foobar();
    void foobar( a, b ) int a; char b; { ; }
```

This informs the compiler that any attempt to use the returned value thefunction 'foobar()' is a mistake and

Real World OS-9



by Andy Jacobson

Intro by Colin McKay

First, let me introduce myself. My name i Colin McKay, and I am the Executive Vice President of the OS-9 Users Group, Inc.

I write a regular column for the Users Group newsletter, the MOTD. The column, "Straight from the Horse's Mouth", is about the use of OS-9 in Educational, Industrial and Scientific institutions. The followin article is a reprint from that column.

Opinions expressed herein are those of the author, and may or may not reflect those of the OS-9 User Group, or the OS9 Underground.

Dept. Pharmacology / Div. Nuclear Medicine an Biophysics UCLA School of Medicine

Andy Jacobson is a graduate student in Pharmacology at the University of California in Los Angeles (UCLA) There, he has become a jack-of-all-trades, and has be come involved in cell biology, organic chemistry, physical chemistry, too much microscopy and scientific visualization, and some C programming. Pretty far from classical pharmacology, though he does use a lot of pharmacological techniques in his work.

Although he does not come from a computer science background, Andy is involved with setting up the system we will be talking about, administers a Sun, and work with several other computer systems.

One of the systems that they use is a Leica Lasertechnik Confocal Laser Scanning Microscope (CLSM), which uses

the run-time version of Professional OS-9 v2.4.

Confocal is a light microscopy technique which differs from standard optical microscopes that we are all familiar with. This method allows somewhat higher resolution and reduced depth of field by focusing light onto (and from) a single point in the sample at a time, allowing for 3-space localization of signal. The signal is recorded in a raster scan, which is where OS-9 comes into the picture, to control the scan mirrors, and collect the data.

The CLSM is used rather than an Electron Microscope for a couple of reasons. With a confocal microscope you can use:

a) whole cells, not just slices;
b) live cells, to do dynamic (time) studies;
c) multiple labels (input and fluorescent light wavelengths can be filtered).

A project currently underway using the CLSM involves looking at models of how chemical receptors produced by cells are regulated, to better understand how they are affected in Parkinson's disease. There are many people using similar techniques to study new drug development and environmental effects on cells.

The CLSM is used examine intracellular localization of dopamine receptors transfected into a mouse fibroblast cell line. Now for the English translation:

Dopamine is an amino acid, and is one of many chemicals (neurotransmitters) which transmit signals in the brain. Improper use of dopamine by cells certain portions of the brain is related to Parkinson's Disease. The recent movie 'Awakenings' dealt with a similar subject.

Genes that produce dopamine receptors are spliced into the the chromosomes of mouse fibroblast cells. The gene is from rat brain, while the host cells are mouse fibroblasts (sort of undifferentiated connective tissue cells), which have no receptors of their own.

Byte length

The ANSI C Standard requires a byte to be at least eight bits in length and integers to be at least sixteen bits, (four bytes), long. The Standard also requires integers to be the 'natural size', or the number of bits the CPU uses to handle a single instruction.

This requirement places the ANSI C Standard in conflict with the older 8-bit CPU processors as the 'natural size' of an integer is usually two bytes in length for these systems. When this conflict arises, the 'natural size' takes precedent over the 8-bit byte requirement.

Integral data type ranges

The compiler dictates certain constraints on the programs it will compile. Translation and numerical limitations represent how the compiler will interrupt typed data objects and constructs. ANSI C compilers must support specified minimum values, but are free to exceed the maximum limitations.

Numerical limits are the range of values of each scalar object type an ANSI C compiler must support and these are defined in the 'limits.h' and 'float.h' include header files.

ANSI C compilers must support the following translation limits:

- 15 nested levels of iteration and control structures, (compound statements).
- 6 nested levels in conditional compilation.
- 12 pointer, array, and function basic-type modifiers in the declaration.
- 127 parentheses-nested expressions.
- 31 significant characters in macro names and internal identifiers.
- 6 significant character in an external identifier.
- 511 external identifiers in one source file.
- 127 identifiers within the scope of one program or function block.
- 1024 macro names within one source file.
- 31 parameters in one function call or definition.
- 31 parameters in one macro definition.
- 509 characters in a source line.
- 509 characters in a string literal (after concatenation).
- 32767 bytes in an array or structure array.
- 8 nested levels for '#include' files.
- 257 'case' labels in a 'switch' statement.

TYPE SPECIFIERS AND QUALIFIERS

Type specifiers provide information or definitions about the data types of the identifier names. The side effects may occur when data types are defined as type tags, structure and union component names, and enumeration constants.

Declaration syntax requires a storage class specifier, a type specifier, a type qualifier, or some combination of these three. Older compilers often did not implement type specifiers, or omitted the 'void' type qualifier and maintenance and readability of source code suffered.

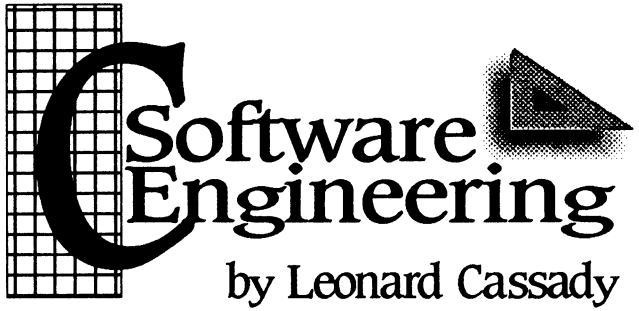
Generally, type specifiers indicate the storage class of return types in functions and denote the various possible side effects of operations on variables. The default type specifier for variables and functions is an 'int' type cast unless otherwise 'specified'.

ANSI C introduced the type qualifiers, 'const' and 'volatile' as part of th standard language. They designate data objects whose values are unchanging and whose value may be altered in ways unknown to the compiler.

'const' qualifier

The 'const' type qualifier specifies a variable that may NOT be modified in anyway once it has been initialized and may be used in place of the preprocessor '#define' directive to identify read-only data.

The 'const' qualifier can produce subtle side effects, depending on its appearance in the definition.



by Leonard Cassady

ANSI C

Last installment, we discussed source translation differences and additions, (lexical analysis), between Traditional C and the ANSI C implementations. This issue, we continue the discussion with data type differences. Next issue we will begin the actual construction of the ANSI Standard Library and the contents of the include header files.

IDENTIFIER NAMES

One of the most important features of any high-lev programming language is the capacity to classify data type There are eight identifier names that may be declared in th C Language.

- preprocessor macros
- statement labels
- type qualifiers
- type tags
- variables
- functions
- enumeration constants
- structure and union components

Preprocessor macros are declared by the '#defin preprocessor directive, and statement labels are declared t their appearance in a C function or source file. Type tag structure and union components, and enumeration constants are 'type specifiers' in declarations. Variables, functions ar types are 'declarators' that appear in declarations.

It then becomes possible to study what the dopamine receptor does on its own, without having to worry about all the other receptor systems in the typical neuronal cell.

The receptors are marked with fluorescent dyes that tend to stain the receptors, and leave the rest of the cell structure relatively unmarked. Also used are other fluorochromes that label cell membranes and other organelles (specialized areas within a cell, roughly analogous to organs), which provide a reference for the location of the receptors.

To make changes to the run-time OS-9 software that is provided by Leica, several packages have been added. From Microware, the Pharmacology Department has added the development pak, NFS (which they found inadequate for their needs), and SrcDbg, as well as mshell. A lot of the software from the TOP package, as well as some of the EFFO tools are also used. GCC may also be added in the future.

Changes made were primarily to give control of the login banner, add an encrypted password file, some utmp functions, etc. The Bourne shell, 'sh', is used for almost everything, along with a number of the utilities from Cabre Microware's umacs is used as the editor.

A modified ftpdc from the good folks at CERN that uses encrypted passwords is also used.

Some of the areas that they still hope to change include:

- expanded user/group id size (for Unix compatability)
- something that really works like tset that will talk to telnet clients for terminal setup.
- a complete csh
- a new Network File Server (NFS) that better meets their needs
- something like colored memory for swap
- Mac floppy support.

(If you can help out in any of these areas, contact me, and I'll pass on your suggestion or address.)

Hardware used is a MC6803 (MVME147, w/MVME147SA-2). There's also an IPP-1 and APAL board from Eltec, some more memory, a Fujitsu 1G dis with a custom driver. Also used are some VME cards from Leica, which run th scanning, Analog to Digital Converte (ADC) and video.

AVS is used for data visualization, on a Sparc10, with some customized soft ware. Animation work is done on th rendering, and transferred to video tape via an Abekas (pronounced abacus). This is essentially a fast hard disk, used in video production to dump frames to video tape in real time.

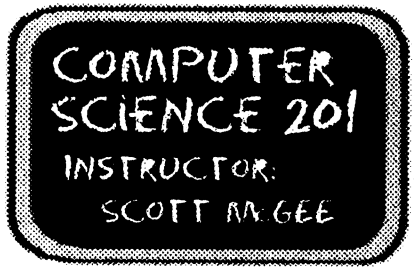
In addition to the confocal micrc scope, PET (Positron Em Tomography) is also used, although that is another story. The department is also involved in the design and synthesis of a new series of environmentally-sensitiv fluorophores.

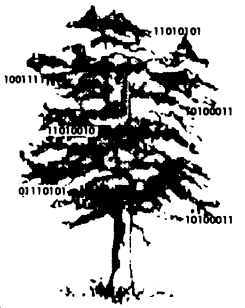
Upcoming columns in future issues of the MOTD include articles about th use of OS-9 by NASA, the Canadia Space Agency, and MIT. MOTD is published 4-6 times per year.

- Colin McKay

(EOF)

If you are interested in joing th OS-9 Users Group and receivin the MOTD, membership details are as follows:
United States and Canada $25.00 US
All other countries $30.00 US
The OS-9 Users Group, Inc. 615 W. 63rd St. Suite 109 Chicago, IL USA 60638



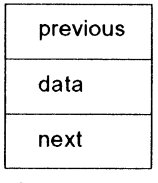


LESSON 4: BINARY TREES

Last time, while discussing doubly linked lists, we used the following definition for a node:

```
typedef struct DLLIST *dllist;
struct DLLIST{
 int    data;
 dllist next;
 dllist previous;
}
```

Graphically, each node would look like this:



As I said last time, we are going to modify this ever so slightly and use it to make an entirely new data structure. In the definition, there are two pointers that point to other nodes (next and previous). We will keep them both, but just rename them. In the following definition, the only other change is to the name we give the types.

```
typedef struct T_NODE *t_node;
struct T_NODE{
 int    data;
 t_node left;
 t_node right;
}
```

Scott McGee can be reached through this magazine at the addresses (Email and regular mail) listed on page 15, or email him directly at "smgee@microware.com"

```
* return true if char is 32..126 and NOT(a..z, A..Z or 0..9)
f_ispunct:  bsr.b f_isprint
        tst.l d0
        beq.b f_ispunct1  not printable, can't
*                         be punctuation!
        bsr.b f_isalnum
        tst.l d0
        bne fail    if it is alphanumeric, it's not punctuation
        moveq #1,d0
f_ispunct1  rts
*******************************************************

* return true if arg is SPACE, TAB, LF, CR, or FF

f_iswhite:  cmpi.l #32,d0  space?
        beq.b f_iswhite1
        cmpi.l #7,d0   tab?
        beq.b f_iswhite1
        cmpi.l #10,d0  lf?
        beq.b f_iswhite1
        cmpi.l #13,d0  cr?
        beq.b f_iswhite1
        cmpi.l #12,d0  ff?
        bne.b fail
f_iswhite1  rts
*******************************************************

#endasm
```

You might have a look at a neat little shortcut in char_conv.c. Instead of creating my own psect for an assembly language file, I have placed a #asm/ #endasm around the assembler code. This way, the compiler takes care of creating the psect instructions.

All the routines in char_conv.c are called in the same way as the equivalent standard functions. It is just a matter of inserting a 'f_' in front of them.

Next month we'll continue on the same theme by presenting some more routines from our library. Don't forget to write...we can be reached at:

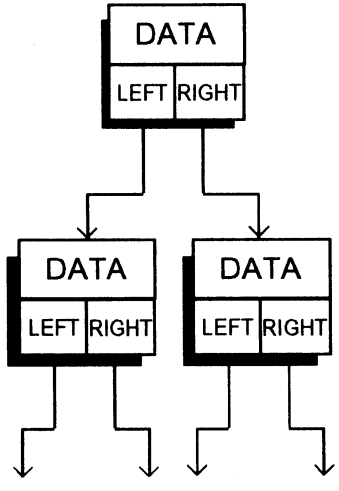
PO Box 355,
Porthill, ID 83853 or

PO Box 57,
Wynndel, BC, Canada V0B 2N0 or
Compuserve :76510,2203.

-Bob van der Poel

(LOF)

Now, you are no doubt wondering why bother changing next and previous to left and right. Well, the reason is, we are not going to have one pointing back to the node that points to us, but will point to ***TWO*** new nodes, instead of one. This way, instead of creating a one dimensional list, we get a two dimensional tree. Graphically, it would look like this:



To discuss trees, we need to define some terms. One that I have already used is *NODE*. A node is one element in the tree. A node may either be **NULL** (*an empty node*) or will be a **T_NODE** structure as defined above. Next is *ROOT*.

The root of a tree is the "first" node in the tree, much like the first nod in a linked list is the head. The next two terms, *PARENT* and *CHILD* are used to define the relationship between nodes. In the diagram above, the top (*or root*) node is the parent node of both of the lower (*or child*) nodes. The root has no parent. The two lower nodes are both child nodes of the root.

We can differentiate between them by calling them the left child and the right child. (*Now you know why we call the pointers left and right.*) If two nodes share the same parent, then they are called *SIBLING* nodes.

Another useful definition is LEVEL, this refers to the depth of the tree. For instance, in the tree shown above, there are two levels displayed. We can also use level to indicate the depth of a particular node. The convention I will use is to call the root node level 1. Its children will be level 2, and so on. A *LEAF* or *TERMINAL NODE* is a node in the tree that has NO children.

An *INTERIOR* node is one that **DOES** have children (since it occurs in the interior of the tree. A node and all the nodes below it is called either a *BRANCH* or a *SUBTREE*.

Ok, now that we agree on some terminology, lets continue with our discussion. Lets talk a little about using trees before we consider any code. Like a linked list, a tree may be random, or it may be ordered. If the tree is ordered, due to its more complex structure, there are a **LOT** more ways to order it. A list is sorted in either ascending or descending order. A tree, however, is not linear, and so does not have to have a linear order like that. There is one type of tree, the **Binary Sort Tree**, that does have such an order. Other types of tree structures, however, may take on other types of ordering schemes. In fact, while the tree structure we defined has two children for each node (*hence, it is called a binary tree*), there is no reason that a tree couldn't have more. In fact, one common implementation is to have a structure like this:

```
typedef struct T2_NODE *t2_node;
struct T2_NODE{
 int   data;
 t2_node sibling;
 t2_node child;
}
```

Continued Page 15

```
#asm
*******************************************************
* Convert character to upper/lower case
f_toupper: move.l d0,-(sp)
        bsr.b f_islower
        tst.b d0
        beq.b f_tolower1  not a lowercase to
start with, do nothing
        andi.b #$df,3(sp)
        bra.b f_tolower1
f_tolower: move.l d0,-(sp)
        bsr.b f_isupper
        tst.b d0
        beq.b f_tolower1
        ori.b #$20,3(sp)
f_tolower1 move.l (sp)+,d0
        rts
*******************************************************
* These are fast versions of isupper() and
* islower(), etc.. The standard library uses a
* table to do this inline. However,  these
* routines are faster, and  shorter. Also, they
* work for unsigned char, which the std.
* library does not:
*******************************************************

*******************************************************
* return true if arg is a..z or A..Z
f_isalpha:  bsr.b f_tolower   convert to
lowercase
        bra.b f_islower   see if lowercase
*******************************************************
* return true if arg is 0..9
f_isdigit:  cmpi.l #'0',d0     see if in range 0..9
        blo.b fail
        cmpi.l #'9',d0
        bra.b f_islower1    branch to "bhi.b fail"
instruction
*******************************************************
* return true if arg is 127 or 0..31
f_isctrl:  cmpi.l #127,d0
        beq.b f_isxdigit3   to RTS
        cmpi.l #31,d0
        bra.b f_islower1    branch to 'bhi fail'
instruction
*******************************************************
* return true if arg is 32..126
f_isprint:  cmpi.l #32,d0
        blo.b fail
        cmpi.l #126,d0
        bra.b f_islower1    branch to "bhi.b fail"
instruction
*******************************************************
* return true if arg is A..Z
f_isupper:  cmpi.l #'A',d0   if in range A..Z
return !0
        blo.b fail      too low, return 0
        cmpi.l #'Z',d0
        bra.b f_islower1
f_islower:  cmpi.l #'a',d0
        blo.b fail
        cmpi.l #'z',d0
f_islower1  bhi.b fail
        rts          return non-zero value (true)
fail    moveq #0,d0     return 0 value (false)
        rts
*******************************************************
* return true if arg is a..z, A..Z, 0..9
f_isalnum:  move.l d0,-(sp)
        bsr.b f_isdigit    see if digit 0..9
        tst.b d0
        bne.b f_isxdigit2   is digit, exit (clean stack!!)
        move.l (sp)+,d0    get old d0 back
        bra.b f_isalpha     and check to see if it's alpha
*******************************************************
* return true if arg is 0..9, a..f, A..F
f_isxdigit: move.l d0,-(sp)     save copy of d0
        bsr.b f_isdigit    see if it's 0..9
        tst.b d0
        bne.b f_isxdigit2   it is 0..9, exit
        move.l (sp),d0     get original d0
        bsr.w f_toupper     convert to
uppercase
        cmpi.l #'A',d0     see if A..Z
        blo.b f_isxdigit1   nope, return 0
        cmpi.l #'F',d0
        bls.b f_isxdigit2   return true
f_isxdigit1 bsr.b fail
f_isxdigit2 addq.l #4,sp       restore stack
f_isxdigit3 rts
*******************************************************
```

not, drop a note and we'll see about doing a series on using make. In addition to the files, I also have an #include file, "bvlib.h" which contains the needed declarations for my functions. I usually #include this file in all my programs.

```
/* bvlib.h */

/* These are routines from "case_conv.c".
They replace the standard stuff
from ctypes.h.  These are faster and are
functions, (not macros).  Plus,
they work with any integer arguments.  */

int f_toupper(), f_tolower(), f_isalnum(),
f_isalpha(), f_isascii();
int f_isctrl(),  f_islower(), f_isprint(), f_ispunct(),
f_isupper();
int f_isxdigit(),f_iswhite();

/* Additional string functions */

char *strsave(), *strend();

/* memory functions */

void *memchr();

/* Useful macros */

#define dswitch(a,b)
switch((tolower((a))<<8)+tolower((b)))
#define dcase(a,b) case((a<<8)+b)
#define arraysize(a) (sizeof((a))/sizeof((a[0])))
#define offsetof(t, m)  (&(((t *)0)->m))
```

Any program needing my library functions just needs a library specification in the makefile.

For example, in the makefile for one program I have the following lines, (these are excerpts):

```
LFLAGS = -q -l=/dd/lib/termlib.l -l=/dd/lib/bvlib.l -m=16k ...
$(OFILE): $(RFILES) $(CC) $(RFILES) $(LFLAGS)
```

Now, our first module! The "standard" libraries usually contain a set of character conversion and classification routines like: isupper(), islowe toupper(), etc. They are normally implemented as macros. For a lot of programming they are just fine, however, they can cause problems. First, the functions only work for characters in the range 0 to 128 and the special character EOF (-1). If you are writing programs using unsigned characters you can't use the standard functions without a lots of problem s.

Second, the fact that the routines are implemented as macros can create subtle problems.

The module, "char_conv.c", implements all of the standard routines as functions. To make these functions as fast and short as possible, I have done the coding in assembler. However, you could do the same in portable C and lose very little. Frankly, doing this in assembler was a good excuse to learn 68000 assembler.

```
/* char_conv.c */

/* This file contains fast version of the
standard character conversion
routines and the is??() routines. They are
much faster than the table lookup
routines supplied in a standard library. They
also handle characters in the
complete range of integers...the standard
library routines return incorrect
data for characters outside the range 0..127

If you use these routines, don't include
<ctype.h> in your sources, use the
<bvlib.h> file instead...
*/
```


Continued from Page 13

With this type of node structure, sibling is basically a linked list of sibling nodes. If sibling is not NULL, then the node has more siblings and this points to the first of them. We will discuss this type of tree in a future article, but for now, lets return to our binary tree.

Keeping in mind the things we learned from working with linked lists (*special handling for insertion/deletion from the head or tail [root or leaf for a tree], and such*) lets consider insertion into a tree.

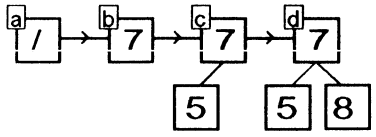
First, we need to decide what sort of ordering (*if any*) our tree should have.

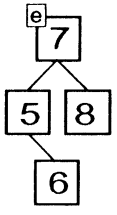
If a node has a variable number of children, it may be very useful without any ordering imposed on it. A directory tree on disk is just such a tree structure. If we are using a binary tree, however, we probably should impose an order on it to aid in locating items within it. I mentioned the **Binary Search Tree** earlier, and we will deal with that now

The ordering that makes a **Binary Search Tree** work goes like this. The value in a node determines which subtree any further values will be in. In other words, if a node contains x as its value, and we are looking for a node that contains y, then it is in one subtree if x<y and the other if y<x. For this particular tree, we will specify that the left subtree contains only values less that that in the node, and the right subtree contains only values that are greater than that in the node. We will reserve discussion of duplicate values for later.

Now, to make sure we understand how we want this to work, lets look an example of inserting into a tree before we

deal with code. We'll start with an empty tree, and insert the following values: 7, 5, 8, and 6. The tree will go through the following changes:





In **a**, the tree empty. In **b**, we insert 7 so it appears at the root. In **c**, we insert 5. Since 5 is less than 7, it goes in the left child. In **d**, we insert 8. 8 is greater than 7 so it goes in the right child. In **e**, we insert 6, which is less than 7 and so goes in the left subtree. Since the left subtree is not null, we look at its root which is 5. 6 is greater than 5 so we insert it in the right child. If we went on to insert 2, it would appear as the left child of 5. Similarly, if we insert 11, it would appear as the right child of 8.

As you can guess from the above sequence, the actual shape of the tree is dependent on both the data it holds and the order in which it was added. You might try building the tree made by adding 5, 7, 8, and 6 to an empty tree.

Now, lets discuss an algorithm to accomplish these insertions.

```
b-tree insert:
  root = tree
  while root != NULL
   if value < root->data
     if root->left != NULL
       root = root->left
     else
       root->left = newNode(value)
       root = NULL
   else
     if root->right != NULL
       root = root->right
     else
       root->right = newNode(value)
       root = NULL
```

What we are doing here is to assign the variable root to the root of the tree. From now on, it will point to the sub tree in which we know the value belongs. The while loop will step root down the tree, branching either left or right as appropriate, until a null node is found. This node, since it is empty, must be the correct location for the value, so a new node is created with value as its data and is assigned there. Finally, root is assigned the value NULL to terminate the loop.

Notice that if tree is NULL, nothing will happen here. This means that we will have to add code to detect an empty tree and assign the root node.

This will be the homework assignment for this session, along with converting the pseudo code above to functioning code. Next time, we will discuss deletion from the tree and try to figure out what to do with duplicate values.

Happy New Year!
Class Dismissed.

-Scott McGee

(EOF)

Building Your Own C Libraries - Part 2

by Bob van der Poel

Last month we started looking at C libraries. If you've forgotten our discussion you might want to do a quick review since we are going to get right into the programming stuff! As you develop programs you will find that you are using the same routines over and over again. Of course, you could just grab the source code for these routines and compile it with your program...but placing it all in a library is much easier. You could create a library of highly specialized routines, (perhaps for statistics, data bases, graphics, etc), or you can just place a number of your own functions into a personal library. This is what I've done.

I use the "make" utility to maintain my library. Following is a copy of my "makefile".

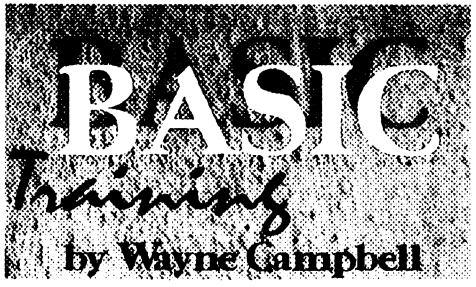
```
#Makefile for bvlib

TEMP  = /r0
CC    = cc
CFLAGS = -t=$(TEMP) -s -q
OFILE  = /dd/lib/bvlib.l
RDIR   = .RELS

RFILES = memchr.r bsearch.r cwriteln.r creadln.r fgets_nl.r lowstr.r\
 upstr.r rnd.r memcount.r strend.r strftime.r strucmp.r struncmp.r\
 strpad.r strstr.r strsave.r strustr.r ptr_strcmp.r char_conv.r

$(OFILE): $(RFILES)
        chd $(RDIR); merge $(RFILES) >-$(OFILE)
```

This makefile contains the names of the various modules in the library and the specific instructions to create the library, (you'll have to change this a bit for Level II). Hopefully, you are familiar enough with the make utility to understand the file...if



NCode DCode Binary File Translators

Some of you may remember my ConvB routine from Volume 1 Issue 9. It created a Basic09 routine called ConvD that included the binary values of the file specified in DATA statements that would be converted back into binary form when it was run. That routine's main flaw was a size limitation. There is a certain size that you could not go beyond due to the memory limitations of Basic09. Well, I've written a new set of routines that allows you to convert any size binary file to a text file, and convert the text file back to binary code. They are called NCode (for eNcode) and DCode (for Decode).

These routines are relatively fast and should be useful for those who wish to transmit binary data and have no way to transmit the data directly as binary code. There are a few caveats however, and the remainder of this article will be devoted to them.

NCode

NCode is passed a single parameter; the pathlist of the binary file to be converted to text. The file may be anywhere on your disks (so long as the disk exists for floppy us When run, any pathlist is removed from the specification for the purposes of creating the output file, as this file is created in your current data directory. The filename portion of the specification is used for the filename to create, with extension '.ND' appended to it.

Continued Page 20


```
0147      ENDIF
0149    UNTIL SUBSTR("/",outfile)=0
0158    PRINT outfile
015D    OPEN #inpath,filename:READ
0169    inopen:=TRUE
016F    CREATE #outpath,outfile:WRITE
017B    outopen:=TRUE
0181    WHILE NOT(EOF(#inpath)) DO
018C     READ #inpath,line
0196     FOR counter:=1 TO LEN(line)-1
          STEP 2
01B0      char:=VAL("$"+MID$(line
          ,counter,2))
01C4      PUT #outpath,char
01CE      PRINT USING "h2",char;
01DA     NEXT counter
01E5     PRINT
01E7    ENDWHILE
01EB 10 IF inopen THEN
01F7     CLOSE #inpath
01FD    ENDIF
01FF    IF outopen THEN
0208     CLOSE #outpath
020E    ENDIF
```

Continued from Page 20

```
PROCEDURE NCode
0000      PARAM infile:STRING[255]
000C      DIM inpath,outpath,char,counter
           :BYTE
001F      DIM inopen,outopen:BOOLEAN
002A      DIM outfile:STRING[255]
0036      inopen:=FALSE
003C      outopen:=FALSE
0042      ON ERROR GOTO 10
0048      counter:=0
004F      outfile:=infile+".ND"
005D      REPEAT
005F       IF SUBSTR("/",outfile)>0 THEN
006F        outfile:=MID$(outfile,SUBSTR
             ("/",outfile)+1,LEN(outfile) )
0086       ENDIF
0088      UNTIL SUBSTR("/",outfile)=0
0097      PRINT outfile
009C      OPEN #inpath,infile:READ
00A8      inopen:=TRUE
00AE      CREATE #outpath,outfile:WRITE
00BA      outopen:=TRUE
00C0      WHILE NOT(EOF(#inpath)) DO
00CB       GET #inpath,char
00D5       counter:=counter+1
00E0       PRINT #outpath USING "h2",char;
00F0       PRINT USING "h2",char;
00FC       IF counter=22 THEN
0108        PRINT #outpath
010E        PRINT
0110        counter:=0
0117       ENDIF
0119      ENDWHILE
011D      IF counter>0 THEN
0129       PRINT #outpath
012F       PRINT
0131      ENDIF
0133 10   IF inopen THEN
013F       CLOSE #inpath
0145      ENDIF
0147      IF outopen THEN
0150       CLOSE #outpath
0156      ENDIF
0158      END

PROCEDURE DCode
0000      PARAM infile:STRING[255]
000C      DIM inpath,outpath,char:BYTE
001B      DIM counter:INTEGER
0022      DIM ext:STRING[3]
002E      DIM line:STRING[44]
003A      DIM filename,outfile:STRING[255]
004A      DIM inopen,outopen:BOOLEAN
0055      inopen:=FALSE
005B      outopen:=FALSE
0061      ON ERROR GOTO 10
0067      ext:=RIGHT$(infile,3)
0072      FOR counter:=1 TO 3
0082       IF MID$(ext,counter,1)="n" OR
            MID$(ext,counter,1)="d" THEN
00A3        ext:=LEFT$(ext,counter-1)
             +CHR$(ASC(MID$(ext,counter,1
             ))-32)+RIGHT$(ext,LEN(ext)
             -counter)
00CE       ENDIF
00D0      NEXT counter
00DB      IF ext=".ND" THEN
00EA       outfile:=LEFT$(infile,LEN
            (infile)-3)
00FA       filename:=infile
0102      ELSE
0106       outfile:=infile
010E       filename:=infile+".ND"
011C      ENDIF
011E      REPEAT
0120       IF SUBSTR("/",outfile)>0 THEN
0130        outfile:=MID$(outfile,SUBSTR
             ("/",outfile)+1,LEN(outfile))
```

As an example:

```
ncode /dd/sys/stdfonts
```

will look for the file stdfonts in the /dd/sys directory and will create a file named stdfonts.ND in the current data directory.

Also:

```
ncode stdfonts
```

will look for the file stdfonts in the current data directory and will create a file named stdfonts.ND in the current data directory. If the file specified does not exist, NCode will end *without* reporting the error.

The file created will contain strings of text that are 44 characters in length (22 bytes of data translated into text). The value of each byte of binary code is converted to text form in hexadecimal format. This makes for a uniform file that is easily transmitted elsewhere, and is easily converted back into binary code. If the last line of the file contains less than 22 bytes worth of data, it will be less than 44 characters in length.

DCode

DCode is passed a single parameter; the pathlist of the text file to be converted to binary. The file may be anywhere on your disks (so long as the disk exists for floppy users). When run, any pathlist is removed from the specification for the purposes of creating the output file, as this file is created in your current data directory. The filename portion of the specification is used for the filename to create. If the extension '.ND' is not present (upper or lower case) it is appended to the filename to find.

As an example:

```
dcode /dd/binaries/stdfonts
```

will look for a file named stdfonts.ND in the /dd/binaries directory and will create a file named stdfonts in the current data directory.

Also:

```
dcode stdfonts.nd
```

will look for a file named stdfonts.nd in the current data directory and will create a file named stdfonts in the current data directory. If the file specified does not exist, DCode will end *without* reporting the error. The file created will be a duplicate of the original binary file.

Additionally, these routines may be used to translate executable files. The file created by DCode will also be executable, but the attributes of the file will not be set for execution. You'll have to set the attributes to be able to execute the file.

NCode and DCode *do not* report errors. You'll want to check the files created when NCode or DCode are run to verify their integrity. These routines may be altered to remedy this, if you so choose, but I would like all changes, additions or alterations to be reported to me via this magazine, or through email to me. My email address is:

zog!wayne@abode.ttank.com

(or use one of the addresses on page 15)

Enjoy!

Continued Page 22