OS-9 Newsletter

Volume III No. 4 | Bellingham OS-9 Users Group | April 30, 1992

Basic09 Part-7
Tutorial

by Scott Honaker & Rodger Alexander

Well, as promised, here is the ***SEARCH*** routines for our ***PDS Database***. I've divided it into two procedures and you could easily simply the process by restricting the field choices and eliminating the first procedure altogether. I did not go to the trouble of fancy screen formatting, however I thought it might be a nice touch to not clear the screen with a "PRINT CHR$(12)" before listing the field options, but instead print a block of spaces to blank out the menu options and place the field choices at that location, leaving the current database displayed in it's original position. Then when you search routine find the desired record, it would simply replace the current record at the top of the screen. BUT, I'll leave those options to your creativity and programming ability.

HOW IT WORKS:
When you select "<F>ind" from the Main Menu, the PDS Procedure runs ***"Search_DB(DB_Path,Top)"***. (This is assuming you have already opened a file and have it displayed at the top left corner of the Main Menu screen.).

LISTING 1:
The PARAMeter line assigns the proper variable types to **DB_Path** and **Top** variables, which were passed to the procedure from the Main Menu. The DIMension statement defines and assigns more fields that will be used in the listing. **CHR$(12)** clears the screen and the next 11 lines print to the screen. The **field** variable contains the number 1 to 8 from the input statement and will determine which variable from the **DATA** field will be placed in the variable **"FieldName"**. So if you selected "2) Last Name" as the field that you want the search routine to base it's search on, then the value of "2" will be used as the top of the "FOR X = 1 TO field" (for x = 1 to 2). and the second variable in the DATA field (Last Name) will be used for the variable **FieldName**.

The whole point of the ***Search_DB*** Procedure is to provide two more variables ***(field and FieldName)*** to be passed along to the actual ***Search*** Procedure. Now we have the name of the field we want to search for (for convenience sake) and the field number, which is necessary for determining field locations and lengths in are record search.

LISTING 2:
The first line defines our ***address*** variable again with each field length and type specified for our database. The second line defines the variable types passed to the procedure from the last line of LISTING 1. The next three **DIM**ension statements could have been combined but were separated in this case for clarification. The first **DIM** line identifies the string variables and their length, the second identifies the byte variables and the third identifies **rec** as our database record variable defined by the **TYPE** statement in the first line of the listing.

The following **PRINT** statements are obvious, and the **INPUT** simply assigns your query to the **SearchString** variable. What follows is not so obvious, so look closely at the comments between each execution line for an explanation of how the database records are searched at specific string locations and those strings are then compared to the **SearchString.**

When the **SearchString** is found, the record it is found in is printed to the screen. A "Press AnyKey" prompt is displayed. At this point in the listing, the **INKEY$** command is called up in the middle of a "WHILE-DO-ENDWHILE" nested routine. When a key press is detected, the *Search* Procedure is ended and we are returned to the Main Menu.

NEXT MONTH, our final chapter (Didn't I say that last month) will feature the **Pack** Procedure.

-- IN THIS ISSUE --

LISTING 1

```
PROCEDURE Search_DB
REM database Search Routine
PARAM DB_Path:BYTE; Top:INTEGER
DIM field:BYTE; FieldName:STRING[12]
PRINT CHR$(12) \ PRINT  (*Clear Screen
PRINT "Personal Database System - Search Database" \ PRINT
PRINT "Search on which field:" \ PRINT
PRINT "1) First Name"
PRINT "2) Last Name"
PRINT "3) Address 1"
PRINT "4) Address 2"
PRINT "5) City"
PRINT "6) State"
PRINT "7) Zip Code"
PRINT "8) Phone Number" \ PRINT
INPUT "Enter choice: ",field
FOR x=1 TO field
READ FieldName
NEXT x
DATA "First Name","Last Name","Address 1","Address 2",
"City","State"
,"Zip Code","Phone Number"
RUN search(DB_Path,Top,field,FieldName)
```

LISTING 2

```
PROCEDURE Search
TYPE address=FName:STRING[10]; LName:STRING[15]; address1:STRING
[20]; address2:STRING[20]; city:STRING[15]; state:STRING[2]; zip:STRING[10]; phone:STRING[14]
PARAM db_Path:BYTE; Top:INTEGER; field:BYTE; FieldName:STRING[12]
DIM CompareString:STRING[20]; SearchString:STRING[20]; char:STRING[1]
DIM offset:BYTE; length:BYTE
DIM rec:address
char="" (*Used in the INKEYS routine

PRINT CHR$(12) \ PRINT          (*Clear Screen
PRINT "Personal Data`base System - Search Database" \ PRINT
PRINT  \ PRINT
PRINT "Enter "; FieldName; " to find";  \ INPUT SearchString
PRINT
FOR X=1 TO field          (*field = value from SearchDB Procedure
READ offset \ READ length          (*reading data variables in pairs
NEXT X
FOR current=1 TO Top    (*Top = number of records in database
SEEK #db_Path,(current-1)*SIZE(rec)+offset       (*Search each record at a specific field location (offset)
GET #db_Path,CompareString       (*assign  variable at offset location to "CompareString"
CompareString=LEFT$(CompareString,length-offset)  (*Crop "CompareString" var. to the proper length
IF SearchString=CompareString THEN  (*Compare "SearchString" with your query ("CompareString")
SEEK #db_Path,(current-1)*SIZE(rec)    (*Find the beginning of the record with the matching variable
GET #db_Path,rec          (*Retrieve the record in the "address" format
PRINT rec.FName; " "; rec.LName (*print fields to screen
PRINT rec.address1
PRINT rec.address2
PRINT rec.city; ", "; rec.state; "  "; rec.zip
PRINT "Phone: "; rec.phone
PRINT \ PRINT
PRINT "Press Any Key to Continue.. "
WHILE char="" DO        (*INKEYS routine
RUN inkey(char)
ENDWHILE
END
ENDIF
NEXT current
DATA 0,10,25,45,65,80,82,92,106
```

PDS Database on DISK

The complete database is available on 5-1/4 inch 35 track, Single Sided Disk format for $1 (and that includes postage and a fancy disk label). Mail your order to *OS9 Newsletter, 3404 Illinois Lane, Bellingham, WA 98226*

1.2 and 1.4 High Density Floppies on your CoCo

Instruction for modifying the ORIGINAL RADIO SHACK FLOPPY CONTROLLER

The controller MUST be the one with the full sized board, a 1793 controller chip and three adjusting potentiometers. According to the Western Digital manual, the 1773 (used in the newer controllers) CANNOT do high density.

This modification is NOT for the faint of heart or those unexperienced with hardware modifications. If you don't know what "piggyback" means when refering to chips, forget it! This modification requires 32 soldering connections, 18 jumper wires and a lot of patience. Do this on you old spare controller if you can. The old controller needs 12 volts therefore you MUST have a multipak or equivalent. This modification will allow the controller to use either 250 kbs or 500 kbs data transfer rate. This is the difference between the standard 5.25" 360k or 3.5" 720k drives and a 5.25" 1.2 meg or 3.5" 1.4 meg drive.

WHAT YOU NEED:

1 74LS74
1 74LS158
1 3.9k 1/4 watt resistor
1 mini DPDT toggle switch (optional)

Wire for the jumpers. (I recommend standard wire wrap wire as RS carries. This is very important, DO NOT use thick wire. Wire wrap wire is 30 gauge. Just right for these kind of projects.)

The mod will be done so if a mistake is made and you want to abandon it, you can just remove all of the jumpers plug in replacement chips for the ones piggybacked to and you'll be back to where you started. If you want this option, buy an extra 74LS74 and a 74LS221. There are NO trace cuts in this mod. IC pins are left out of the socket to get the equivalent of a trace cut. If you need to reverse the mod, those pins MUST be reinserted into their respective sockets. There is ABSOLUTLY NO GUARANTEE OR WARANTEE EXPRESSED or IMPLIED FOR THIS MODIFICATION. Now, on to the fun part!!

We will be piggybacking a 74LS74 on to the existing 74LS74 at IC1. We will also be piggybacking a 74LS158 onto the 74LS221 at IC7. Some other chips will be soldered to and some pins will be removed from the sockets for some IC's. These instructions will be entirely verbal, no illustrations.

First, remove U1 (74LS74) from it's socket. Position a new 74LS74 on top of it with the pins EXACTLY overlapping (this is called piggybacking). Be sure both pin 1's are lined up or it'll be poof time when you apply the power. On the upper 74LS74, bend up pins 2,3,5,6,8,9,10,11,12 and 13 so they point directly away from the body of the IC. Pins 1,4,7 and 14 should still be overlapping the lower 74LS74. Carefully solder these pairs of pins together being careful not to blob the solder onto the legs of the lower 74LS74 as you will be plugging the pair (stack) of chips back into the U1 socket when done. On the lower 74LS74, bend pin 11 out away from the body of the chip as you did for some of the pins on the upper IC. Pin 11 will NOT be going back into the socket. Prepare six 3" jumper wires (prepare means strip back the insulation on each end of the wire, no more than 1/16". Then tin the exposed wire on each end of the jumper). Solder the wires to the stacked IC's as follows:

One end of each wire will be unconnected.

1 jumper to pin 11 on the lower IC (the pin sticking out)
1 jumper to the lower IC pin 3 (must still be able to go into the socket)
1 jumper to the lower IC pin 6 (must also be able to go back into the socket)
2 jumpers the the upper IC pin 3
1 jumper to the upper IC pin 6

Also, prepare a 1.5" wire and solder it from the upper IC pin 2 to the upper IC pin 6 taking care not to disconnect the wire already on the upper IC pin 6. You may now carefully plug the IC stack back into the IC1 socket making sure all pins get seated into the socket with the exception of pin 11.

Second, we'll be doing a similar piggyback mod to the 74LS221 in the U7 socket Remove the 74LS221 from the socket. Position the 74LS158 on top of the 74LS221 Make sure that the two IC's are properly aligned and that the two pin 1's are aligned together. Bend up all of the pins on the upper IC EXCEPT pins 8 and 16. solder the two pin 16's together and also solder the two pin 8's together. As before, make sure not to blob solder on the legs as the stack will be plugged back into the U7 socket. Bend pin 13 on the lower IC away from the body of the IC so it cannot be reinserted into the socket. Prepare four 1.5" jumpers, one 2" jumper and one 3" jumper. Solder them in as follows:

1 2" jumper to the lower IC pin 2
1 1.5" jumper from the joined pin 8's to the upper IC pin 15
1 1.5" jumper from the upper IC pin 15 to the upper IC pin 10 (taking care to not disconnect the wire already at pin 15)

1 1.5" jumper from the tied together pin 16's to the upper IC pin 11
1 1.5" jumper to the upper IC pin 7
1 3" jumper to the upper IC pin 1

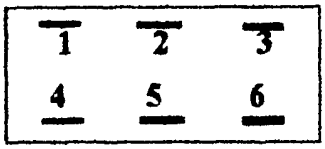
Plug the stack back ino the U7 socket making sure all of the pins are seated firmly EXCEPT pin 13 which should be sticking out. Solder a 3.9k resistor from the upper IC pin 9 to the side of R18 (3.9k) which is the closest to the U7 socket.

Final Assembly

Remove U11 (the 74LS629). Solder one of the 3" jumper wires from U1, the upper 74LS74 pin 3 to the top of the 74LS629 pin 7 making sure not to blob solder. Plug U11 back in making sure ALL of the legs seat firmly into the socket. Unplug U3 (7406 or 7416). Connect the 2" wire from the lower IC pin 2 of the stack at U7 to the top of pin 1 of the IC that was in U3 (making sure not to blob solder on the leg). Plug U3 back into it's socket making sure all of the legs seat firmly into the socket.
Solder the open end of the jumper connected to U1 lower IC pin 11 to U7 upper IC pin 4
Solder the open end of the jumper connected to U1 lower IC pin 3 to U7 upper IC pin 3
Solder the open end of the jumper connected to U1 lower IC pin 6 to U7 upper IC pin 2
Solder the open end of the jumper connected to U1 upper IC pin 6 to U7 upper IC pin 5
Solder the open end of the jumper connected to U1 upper IC pin 3 to U7 upper IC pin 6

Choose 1 of the following select methods:

Select option 1 - using WRITE PRECOMP bit and a SWITCH (For Hard Drive booting systems) Mount the dpdt mini switch somewhere handy. I mounted mine in the hole near C1 and the piggybacked 74LS74's. Make sure that the switch DOESN'T SHORT OUT any traces! I'll refer to the switch pins as follows:



pin 2 toggles between pins 1 & 3
pin 5 toggles between pins 4 & 6

Carefully remove U12 (the 1691)from its socket. Bend up pins 9 and 16 away from the body. Put the 1691 back into the U12 socket making sure that all pins firmly seat with the exceptions of pins 9 & 16.
Prepare and solder a 4" jumper from U12 (1691) pin 9 to the DPDT switch pin 5.
Solder the open end of the jumper connected to U7 upper IC pin 7 to U12 pin 16.
Solder the open end of the jumper connected to U7 upper IC pin 1 to the DPDT switch pin 2.
Prepare and solder a short jumper from the DPDT switch pin 3 to the DPDT switch pin 4.
Prepare and solder a short jumper from the DPDT switch pin 4 (taking care to not disconnect the wire already there) to a convenient ground (for example, IC1 pin 7 on the SOLDER side of the board).
Prepare and solder a short jumper from the DPDT switch pin 1 to the DPDT switch pin 6.
Remove U8 (the MC14174) and prepare a 3.5" jumper. Solder a wire to the top of pin 12 without blobing solder on the leg. Plug U8 back in making sure all of the pins seat firmly into the socket.
Solder the open end of the jumper connected to U8 pin 12 to the DPDT switch pin 1 taking care not to disconnect the wire already there.
Skip to check procedure below.

Select option 2 - using a DRIVE SELECT BIT
Carefully remove U12 (the 1691) from its socket. Bend up pin 16 away from the body. Put the 1691 back into the U12 socket making sure that all pins firmly seat with the exception of pin 16.
Solder the open end of the jumper connected to U7 upper IC pin 7 to U12 pin 16.
Remove U2 (7406) from the socket. Choose a drive select line to use, either DS1 or DS2 (DS0 should not be used or you will not be able to boot, DS3 is usually used to access the back side of double sided drives so that cannot be used either). Solder a 2" jumper to pin 3 (DS1) OR pin 5 (DS2) without blobing solder on the leg. Plug U2 back into it's socket making sure all pins seat firmly.
Solder the open end of the jumper just attached at U2 to U7 Upper IC pin 1.

CHECK PROCEDURE:

Now recheck the entire procedure to make sure no mistakes were made. Check all soldering joints for good connections. Check for shorts, especially by the DPDT switch. There should be NO unconnected jumper wires! If there are, go through the entire sequence to see what you missed. Now, we need to calibrate and test the controller.

Use a multipak which will protect the CPU (you need +12 anyway) in case you made a fatal wiring mistake. Plug the controller into slot 4 as usual. Power on the multipak, then the computer. If the DISK BASIC message doesn't come up quickly then shut the computer off immediatly and power everything off. Unplug the controller and check for shorts and recheck all connections against the modification procedure. If all else fails, you can always remove the piggybacked stacks at U1 and U7, carefully pull off all of the jumpers, insert a new 74LS74 into U1 and a new 74LS221 into U7, pull out U12, carefully bend pins 9 and 16 back down and reinsert it into it's socket, remove the switch and you'll be back to where you started. Presuming you made it past the smoke test, you will need to figure out your switch position and calibrate the controller.

SWITCH POSITION DETERMINATION (Skip if drive select method was chosen)
When the switch is in the position such that pins 1 & 2 are connected together (also pins 4 & 5) the controller is in the HIGH DENSITY enabled position (use a meter to test the connection between pins 1 & 2). When the switch is the other way, the normal configuration is active, which means write precomp is available. Put the switch into normal position for calibration.

CALIBRATION OF THE VCO:
The controller can be calibrated either with a scope or by trial and error. Either way, mark the original position of R8 so you can reverse the modification if you can not get it to work right.

If using a scope, connect the scope to the VCO output of the 74LS629 (U11) pin 7 and adjust R8 for 4 mhz. If doing the adjustment by trial and error, put a formatted RSDOS disk into drive 0 and do a DIR from RSDOS. Turn R8 until you can get a directory. You may have to do lots of DIR commands. Try to find the extreme settings of R8 that will still produce a directory, then set R8 between the two extreme settings. The range in which the DIR will work will be quite small and your final setting for R8 should be as close as possible to the middle of the range. THAT'S IT FOR THE HARDWARE.

To complete the modification you need to apply my IPATCH file "***cc3diskhigh.ipc***" to the ORIGINAL Radio Shack *cc3disk* edition 9 (CRC $759161). You will also need one or both of the following disk descriptors: ***d1_1.2.dd*** (high density 5-1/4 inch drive) and ***d1_1.4.dd***.(high density 3-1/2 inch drive) The patched *cc3disk* detects the old 8inch drive bit in IT.TYP in the drive descriptor and uses it to switch the data transfer rate.

Make a new OS9 boot with the new ***CC3Disk*** and the appropriate drive descriptor(s) to match your high density drive(s). After booting put the switch (option 1) into the HIGH DENSITY ENABLED position and you're ready to go. For use in RSDOS, the switch (option 1) should be in the normal position. Note, the high density drive is not usuable in RSDOS.

P.S. please send any comments or clarifications to the me on the TC^3 UG board. 612-422-0824 Robert E. Brose II 12-1-90

EDITOR'S NOTE: The IPATCH file *(cc3diskhigh.ipc)* and the high density drive descriptors *(d1_1.2.dd* and *d1_1.4.dd)* are all in an archived file called **CC3DIS.AR** found on the *COLUMBIA HTS. BBS* (206-425-5804) and *OS-9 TACOMA BBS* (206-566-8857. Theses are both FidoNET Bulletin Boards and there is no charge for downloading thises files.

pwd and pxd mysteries revealed

OK, so there isn't an OS-9 System Call which returns your current data or execution directory. So, just how do pwd and pxd do it? This document will attempt to explain that so that you will understand it. Also, understanding how pwd and pxd work will also further your knowledge of OS-9 Disk Structure.

Let us begin our discussion with the pwd command, first. Once it's operation is explained and understood, the operation of pxd will be a relatively simple matter to understand.

First off, a small discussion on OS-9 Disk Structure. The basic entity which is contained on any disk, OS-9 ot otherwise is known as a file. Actually, the information stored on a disk is stored in fixed length segments normally referred to as sectors. The operating system involved organizes information stored in these sectors into discernable units. These are the files contained on that disk. Special information is stored by the operating system on the disk which indicate to the operating system how many files there are on a disk, what their names are, where they are located on the disk, and how large they are. This is the basic amount of information which would be required.

OS-9 and other operating systems also store various other information about each file such as the date of creation, date last modified, and so forth. With Disk Extended Color Basic, this information was stored in part of Track 17. In this manner, you are limited to the number of files the disk can store by the amount of space allocated to this special storage area. OS-9 and other operating systems allows for a greater capacity by storing this

"overhead" file information in areas which are dynamically allocated. These areas are normally referred to as "directories" and their related data structures.

A directory is nothing more than a special file manipulated only by the operating system. In the case of OS-9, there is also a special sector allocated to each file to store special information. This sector is called the file descriptor sector. In actuality, with OS-9, the only information stored in the directory file is the name of the files it contains and for each file a pointer to the file's file descriptor sector. File descriptor sector's exist on a one to one correspondence with all files on the disk.

A file's file descriptor segment can be anywhere on the disk, it's actual location is irrelavent in regards to the relative position of the directory which contains the file's name or to the data which the file contains. One special characteristic of the OS-9 disk structure which is shared by several other operating systems is that a directory can contain a directory. This is possible since a directory is nothing more than a file. Therefore a directory file, just as any other file has its very own unique file descriptor sector. This means that the directory structure on a disk can take on a hierarchical structure, a structure which can be compared to a tree. The leaves are the individual non directory files. The branches are the directories. All this has to start somewhere and as in a tree this special directory is called the "root" directory. It is the ultimate parent of all other directories on the disk. The directory file which contains another directory is referred to as the "parent" directory of that directory.

The name of the root directory is the same as the name of the device on which it resides. Thus the name of the root directory on a hard disk who's name is /h0 is /h0. This concept is a little more confusing with a floppy disk drive. If you have two floppy disk drives, /d0 and /d1, then the name of the root directory of a floppy disk which is in device /d0 is /d0, but the name of the root directory of that very same floppy disk when the disk is placed in device /d1 is /d1. Root directories are a little more capable than that of a tree, since it can contain not only directories (branches) but also non-directory files (leaves).

The root directory is automatically created on a disk when the disk is formatted. subsequent directories which are created on the disk are created by the OS-9 System call I$MakDir. A simple user interface to this system call is provided with the Shell command makdir. The first directories created on a disk must first reside in the root directory. Once a directory is created in the root directory, a directory can then be created in that directory. Any or all of these directories can contain non-directory files or other directories.
A lot of this may be review for many readers of this file, but please bear with me I am leading up to the important point here. That is, every directory always contains two special files which are created automatically by the I$MakDir System Call. These are "hidden" files since the Shell dir command doesn't report their existance, but there are there nonetheless. Actually, they aren't really files, but merely special entries in the directory file. This will become apparent shortly.

The names of these entries are "." and "..". The names of these entries do not actually contain the surrounding quotation marks, they names are just the periods. I will surround them with quotation marks to separate them for clarity. These entries could be better referred to as "pointers" to files which already exist. The entry "." is a pointer to the directory itself and the entry ".." is a pointer to the parent directory of the directory. Maybe an example will help to clear up any questions. Let's say we have a disk named /h0. It's root directory's file descriptor sector is located at LSN 25. (I'll keep this simple and use all decimal numbers for LSN, but in actuality LSNs are stored as 3 byte binary integers.)

Now, let's say we create a directory in the root directory called /h0/TEXT. Let's also say that the file descriptor sector for the directory is located at LSN 50. The /h0/TEXT directory will automatically have those two special entries "." and "..". The LSN of the file descriptor sector for the "." entry will be 50 and the LSN for the file descriptor sector for the ".." entry will be 25. An important thing to remember is that the entry for ".." and "." for the root directory are the SAME, because the parent of the root directory is the root directory itself.

These entries are really "synonym" file names. So, if your current data directory is /h0/TEXT and you ask for a directory of file "..", then you are really asking for a directory of /h0. The entries "." and ".." can be used ANYWHERE that a directory name can be used. For example, you can also "chd .." to change your data directory to the parent of the directory you are current in.

Now, just how does pwd make use of this information to determine your current data directory? Well, pwd first opens the directory file ".", the current data directory, and reads the entries ".." and ".". This gives pwd the LSN of the file descriptor sector for the parent directory and the current directory. If these are equal, then pwd has finished it's job, it is now at the root directory. If they are unequal, then we need to determine the name of the "." directory. This is the main processing loop for pwd. The name of the "." directory is done by changing the current data directory to the directory "..". It then checks to see if the LSNs for ".." and "." are equal. If they are, pwd is done. If they are unequal, it reads through the directory searching for an entry whose file descriptor

sector LSN is equal to the file descriptor sector LSN for the previous "." entry. When that is found, we have the name for the previous "." entry. That name saved away. We now have the file descriptor sector LSN for this "." entry, so we start the process over by changing directory to ".." again. When it finally finds a directory whose file descriptor sectors LSN are equal, it determines the name of the device with a I$GetStt SS.DevNm System Call and displays the completed result.

You'll probably have to read the previous paragraph a few times before it becomes clear to you just how this process is done. Once you understand it, it will be obvious to you that to perform the similar pxd command, all that needs to be done is to access the directories with the execution bit set! In fact if you compare the pwd program with the pxd program, there are only 4 bytes which are different! (Not counting the 3 CRC bytes, of course!) And two of those4e four are the internal names of the programs! So actually, there are only two bytes which are effectively different for the two programs and these are the two bytes which control the access mask for the I$Open of the "." directory and the I$ChgDir to the ".." directory.

If you can read C, I have included the equivalent code in C for the pwd command and the pxd command. While it would be extremely inefficient for you to compile and actually use these programs they are included for instructional purposes only. See, the original pwd and pxd commands are written in ASM, which makes their binaries much smaller! If you wish to incorporate the code from these two programs in a program you are writing to avoid forking pwd and/or pxd commands, you are most welcome to do so! They were written in a form as to NOT require any special library, they will compile and work just fine with the stock clib.l which comes with the Microware C Compiler for OS-9/6809. In fact, even though pwd could have been written slightly simpler using the Kreider Lib, pxd would not even be possible, since there is no option with the opendir() function to open the directory with the execution bit set, which is required by the pxd function.

I hope this file and the associated C program sources has helped increase your knowledge and understanding of OS-9 and its disk file structure.

C Tutorial

Chapter 2

Editor's Review: Last month we were introduced to "C". We covered some very basic language syntax and learned what an IDENTIFIER is. We learned how to use the CC1 executable modules and even compiled three short programs.

LETS PRINT NUMBERS

Type in the listing named **ONEINT.C** and display it on the monitor for our first example of how to work with data in a C program.

ONEINT.C:

```
main()
{
int index;
    index = 13;
    printf("The value of the
index is %d\n",index);
    index = 27;
    printf("The value of the index
is %d\n",index);
    index = 10;
    printf("The value of the index
is %d\n",index);
}
```

Listing 4

The entry point "main" should be clear to you by now as well as the beginning brace. The first new thing we encounter is the line containing "int index;", which is used to define an integer variable named "index". The "int" is a reserved word in C, and can therefore not be used for anything else. It defines a variable that can have a value from -32768 to 32767 on most microcomputer implementations of C. Consult your users manual for the exact definition for your compiler. The variable name, "index", can be any name that follows the rules for an identifier and is not one of the reserved words for C. Consult your manual for an exact definition of an identifier for your compiler. The final character on the line, the semi-colon, is the statement terminator used in C.

We will see in a later chapter that additional integers could also be defined on the same line, but we will not complicate the present situation.

Observing the main body of the program, you will notice that there are three statements that assign a value to the variable "index", but only one at a time. The first one assigns the value of 13 to "index", and its value is printed out. (We will see how shortly.) Later, the value of 27 is assigned to "index", and finally 10 is assigned to it, each value being printed out. It should be intuitively clear that "index" is indeed a variable and can store many different values. Please note that many times the words "printed out" are used to mean "displayed on the monitor". You will find that in many cases experienced programmers take this liberty, probably due to the "printf" function being used for monitor display.

HOW DO WE PRINT NUMBERS

To keep our promise, let's return to the "printf" statements for a definition of how they work. Notice that they are all identical and that they all begin just like the "printf" statements we have seen before. The first difference occurs when we come to the % character. This is a special character that signals the output routine to

stop copying characters to the output and do something different, namely output a variable. The % sign is used to signal the start of many different types of variables, but we will restrict ourselves to only one for this example. The character following the % sign is a "d", which signals the output routine to get a decimal value and output it. Where the decimal value comes from will be covered shortly. After the "d", we find the familiar "\n", which is a signal to return the video "carriage", and the closing quotation mark. All of the characters between the quotation marks define the pattern of data to be output by this statement, and after the pattern, there is a comma followed by the variable name "index". This is where the "printf" statement gets the decimal value which it will output because of the "%d" we saw earlier. We could add more "%d" output field descriptors within the brackets and more variables following the description to cause more data to be printed with one statement. Keep in mind however, that it is important that the number of field descriptors and the number of variable definitions must be the same or the runtime system will get confused and probably quit with a runtime error. Much more will be covered at a later time on all aspects of input and output formatting. A reasonably good grasp of this topic is necessary in order to understand the following lessons. It is not necessary to understand everything about output formatting at this time, only a fair understanding of the basics.

Compile and run ONEINT.C and observe the output.

HOW DO WE ADD COMMENTS IN C

Load the file COMMENTS.C and observe it on your monitor for an example of how comments can be added to a C program.

COMMENTS.

```
 /* This is a comment ignored by
the compiler */

main() /* This is another comment
ignored by the compiler */
{
   printf("We are looking at how
comments are "); /* A comment is
  allowed to be
  continued on
  another line */
   printf("used in C.\n");
}
/* one more comment for effect
*/
```

Listing 5

Comments are added to make a program more readable to you but the compiler must ignore the comments. The slash star combination is used in C for comment delimiters. They are illustrated in the program at hand. Please note that the program does not illustrate good commenting practice, but is intended to illustrate where comments can go in a program. It is a very sloppy looking program.

The first slash star combination introduces the first comment and the star slash at the end of the first line terminates this comment. Note that this comment is prior to the beginning of the program illustrating that a comment can precede the program itself. Good programming practice would include a comment prior to the program with a short introductory description of the program. The next comment is after the "main()" program entry point and prior to the opening brace for the program code itself.

The third comment starts after the first executable statement and continues for four lines. This is perfectly legal because a comment can continue for as many lines as desired until it is terminated. Note carefully that if anything were included in the blank spaces to the left of the three continuation lines of the comment, it would be part of the comment and would not be compiled. The last comment is located following the completion of the program, illustrating that comments can go nearly anywhere in a C program. Experiment with this program by adding comments in other places to see what will happen. Comment out one of the printf statements by putting comment delimiters both before and after it and see that it does not get printed out.

Comments are very important in any programming language because you will soon forget what you did and why you did it. It will be much easier to modify or fix a well commented program a year from now than one with few or no comments. You will very quickly develop your own personal style of commenting.

Some compilers allow you to "nest" comments which can be very handy if you need to "comment out" a section of code during debugging. Check your compiler documentation for the availability of this feature with you particular compiler. Compile and run COMMENTS.C at this time.

GOOD FORMATTING STYLE

Load the file GOODFORM.C and observe it on your monitor. It is an example of a well formatted program.

GOODFORM.C

```
main()   /* Main program starts
here */
{
   printf("Good form ");
   printf          ("can aid in
");
   printf
   ("understanding a
program.\n");
   printf("And bad form ");
   printf                ("can make
a program ");
   printf
         ("unreadable.\n");
}
```

Listing 6

Even though it is very short and therefore does very little, it is very easy to see at a glance what it does. With the experience you have already gained in this tutorial, you should be able to very quickly grasp the meaning of the program in it's entirety. Your C compiler ignores all extra spaces and all carriage returns giving you considerable freedom concerning how you format your program. Indenting and adding spaces is entirely up to you and is a matter of personal taste. Compile and run the program to see if it does what you expect it to do.

Now load and display the program UGLYFORM.C and observe it. How long will it take you to figure out what this program will do?

UGLYFORM.C

```
main()    /* Main program starts
here */(printf("Good form
"),printf
  ("can aid in
"),printf("understanding a
program.\n")
,printf("And bad form
"),printf("can make a program ");
printf("unreadable.\n"),)
```

Listing 7

It doesn't matter to the compiler which format style you use, but it will matter to you when you try to debug your program. Compile this program and run it. You may be surprised to find that it is the same program as the last one, except for the formatting. Don't get too worried about formatting style yet. You will have plenty of time to develop a style of your own as you learn the language. Be observant of styles as you see C programs in magazines, books, and other publications.

This should pretty well cover the basic concepts of programming in C, but as there are many other things to learn, we will forge ahead to additional program structure.

PROGRAMMING EXERCISES

1. Write a program to display your name on the monitor.
2. Modify the program to display your address and phone number on separate lines by adding two additional "printf" statements.

NW CoCo Fest UPDATE

The Best Western Bayview Inn. in Bremerton is offering us a special group rate. They have set aside 20 rooms for CoCo nuts and their companions until the end of May. A single room is only $55. A double is $58. To get these special rates, contact:

"Betty", Groups Coordinator
Best Western Bayview Inn
CoCo Fest II Convention
5640 Kitsap Way
Bremerton, WA 98312
1-800-422-5017 or (206) 373-7349
Fax: (206) 377-8529

Besides the nicely appointed rooms there is an indoor pool and hot tub. Several couples mentioned how relaxing that was last year. There is a park for walking (a morning run) just two blocks away. There are also discount tickets for the gym that is right next door to the Bayview. Of course, there is a lounge and fine restaurant right in the Inn for those who want to stock up on calories instead of deplete them.

Bremerton is the home of the Puget Sound Naval Ship Yard and again the home of the battleship USS Missouri. It is only about 10-15 minutes from the CoCo Fest II Conference site. Let Betty know that you are interested in the tourist attractions when you call or write.

Bob van der Poel is the first major speaker to confirm. He will attend the entire event. He will be available for personal chats, be our keynote speaker, and will give a special presentation about the language C. Bob is the author of the highly respected and used Ved Text Editor, Vprint Text Formatter for OS9, and the every popular Telewriter word processor for the CoCo.

Finally, the planning people want to let you know that there is a second in the series of CoCo Fest mugs coming. A new design, a new collectable. With your registration they are only $5 apiece, $25 for a set of six. The mugs will be $6 at the Fest. There will also be a fabulous T-shirt and even more! We don't want to give away all the surprises!

Club Activity Report

Bellingham OS9 Users Group - Longview CoCo Club
Mt. Rainier CoCo Club - Port O'CoCo Club - Seattle 68xx Mug

Bellingham OS-9 Users Group

The First Official Meeting of 1992 was held at the Bellingham Public Library on Wednesday, April 22. The room was perfect for our needs with a black board, video tapes, overhead projectors, large table, etc. Unfortunately we were very few in numbers. Only 5 were in attendance. But this was only the first meeting and it takes time to get the word out. Right?!

First order of business was to have each attendee introduce themselves and tell the others about their Computer systems and what they wanted to gain from the club. It was interesting that everyone had several CoCo systems and and IBM Clone.

Northwest CoCo Festival

JUNE 26-27 PORT ORCHARD, WASHINGTON

REGISTRATION FORM

NAME:_____

MAILING ADDRESS:_____

CITY:_____ST/PROV:_____ZIP/POSTAL CODE: _____

EVENT	INCLUDED	SEPARATELY
FRIDAY EVENING		
7:30p.m. Notable Video Tape Presentations	YES	$ 3.00
9:00p.m. Public Domain/Shareware Swap	YES	$ 3.00
SATURDAY:		
7:30a.m. Saturday Swap Meet setup time	n/a	n/a
8:15a.m. No Host Breakfast about	$ 5.00	$ 6.00
9:00a.m. Computer Swap Meet	YES	$ 3.00
10:30a.m. Preliminary speakers	YES	$ 3.00
(Speakers to be announced)		
12noon Luncheon & Keynote Speaker	YES	$12.00
-- featuring Bob van der Poel --		

1:30p.m. Workshops YES $ 3.00
Session 1: to be announced Session 2: to be announced
3:00p.m. Workshops YES $ 3.00
Session 3: to be announced Session 4: to be announced

REGISTRATION FEE:	$20.00	$30.00
(Remember, Saturday No Host Breakfast $5/$6 extra)		

Northwest CoCo Festival Items Available	Order Now	At Fest II
CoCo Fest II Mugs	$ 5.00	$ 6.00
Set of Six	$25.00	$30.00
T-Shirt (Men's M L EXL)	$ 8.00	$10.00
Extra Extra Large add $2.00	Your Name as part of design add $2.00	

Desired Name(s):_____ (size _____)

Second name:_____ (size _____)

Third name:_____ (size _____)

(All amounts are U.S. Currency only)

Advertisement went out announcing that this was a Color Computer Club, yet everyone was interested in their CoCo because they wanted to learn more about OS9.

Second order of business was to display the club offerings and benefits, from the Newsletter to the Public Domain Library, to technical assistance.

Third order of business was all the recent gossip the *NW CoCo Fest*. We are all very excited about Bob van der Poel, and at least four of are planning on going.

Last order of business was a review of Multivue and the currently available upgrade files and patches. Shell+ was reintroduced and a hard drive boot disk with the Multivue Term and windows was created using Burke & Burke's ***EZGEN***.

Next month a demonstration will be provided by Rodger Alexander on the ways to use **AIF** files to improve the performance of Multivue.

-- Rodger Alexander --

Mt. Rainier CoCo Club

Our April meeting started off with a demonstration by Randy Kirschenmann of his "C" program that will display a text file backwards on the screen. The members stated several uses for this program which included a good exercise in using "C". Randy then showed the source code and explained how each line worked.

Chris Johnson then brought the club up to date on the upcoming N.W. CoCo Fest II. It will be even bigger and more interesting than last years Fest. So clear you calendars for June 26th and 27th and plan to attend.

The rest of the meeting was devoted to open discussions of several subjects. Michael Stokes' series on "C" had to be delayed by a month so he will begin it at the next meeting. It should be great for both the beginner and experienced "C" programmers.

-- Alan Johnson --

Seattle 68xxx Mug

The April meeting was very informative. First on the agenda was a hardware presentation showing how to add a second drive to a Color Computer System. Normally this is no big deal and very easy to do. However, in this case the two drives used were the two worst possible cases. The first drive was a new FD-502. The drive configuration jumpers are located under the circuit board and without any labels. Also the jumper pins are arranged in a 90 degree angle and not in the usual configuration. The jumper positions are parallel to the back edge of the drive rather accross the jumper pins as with most drives. It actually boils down to trial and error. Note the default position of the jumper and them move the jumper down two pins.

The second drive was an original 35 track, single sided drive that had no drive select jumper setting on the circuit board at all. However, three solutions are possible. The first one is the Radio Shack method: Get an old Radio Shack drive cable with the card edge connectors that have the missing contacts. The end connector has it's contacts pulled to configure the drive for Drive1, while the second connector has it's contacts pulled for Drive 0.

The second method is to cut the wires on the cable to the drive disabling the Drive0 and Drive3 circuits.

The third method is to cut the trace lines on the circuit board that connect to the card edge connector, again disabling the Drive0 and Drive3 circuits. We tried this method and it worked!

Next on the agenda was the ***PDS Database***. A sample **SEARCH** Procedure was displayed on an overhead projector. Everyone participated in debugging the sample code until it was determined to be perfect. This is the **SEARCH** procedure featured in this Newsletter. However, it took several more modifications to get it to actaully work properly!!!!!

Donald Zimmerman from the Port O'CoCo Club was present at the meeting and updated us regarding the ***NW CoCo FEST*** in Port Orchard on June 26th and 27th. Donald also was seeking suggestions for improving the format and/or subject matter. Most of those in attendance indicated their intentions to attend.

Finally, Rodger Alexander took orders for a special purchase of 720K floppies and 20Meg Hard Drives. The final tally was 7 floppies @ $20 each, 1 Hard Drive for $85 and 2 Hard Drive controller cards for $12 each.

-- Rodger Alexander --

Port O-CoCo

It was like a fairy tale gone awry. Of course, it had a happy ending. That's a given. It had a cast of notable people. Those stories always do. There was drama, action, disappointment, and goodness overcoming all at the end. Well, that was April's meeting.

There was a plan, a storyline. Guests speakers had been invited, repeatedly. Publicity had been created and delivered to the harolders of the day, the newspapers. The grapevine had been used to spread the word, everyone had been called about this great event.

Now for the drama. The guest speaker was contacted 20 minutes before the meeting to make sure all was well in the kingdom. It wasn't. His new chariot had blown a transmission. Goodness would have it that he had an extended warranty on it (we won't give names here). So the focal point of our meeting had just gone dead in the water. Well, life will continue.

A few people had said that they had equipment they were thinking of liquidating, . . . for a price. You never know if they are going to materialize or not. Two of them

never did. There were three boxes of CoCo "stuff" at the home of one member. The night before the meeting it was discovered that he was out of state because of a family death. Attempts were made to get the stuff to the meeting anyway, maybe.

Seven pm rolled around and it looked like a national meeting of veterans of the Civil War. If we'd been holding hands with each other I would have been hugging myself.

But then then the light began to show on this dismal story. People started wandering in between 7 pm and about 8:15. And the group grew to a respectable 16 people. The CoCo "stuff" arrived. Tom Brooks, in his time of sorrow, had seen that the CoCo stuff collected by the Computer Bank Charity was on its way to the meeting. A young man who had had an extensive system came looking for a way to clean out some of his space by getting rid of his CoCo stuff, several boxes of it. We made arrangements for him to donate everything to the Computer Bank Charity who in turn would allow the Port O' CoCo group to sell it for supporting funds for the charity. The bidding started slowly, very slowly. But a spark occurred and things started picking up. And before it was all said and done $160 was raised for the CBC. And a lot of people held in their clutches hardware, software, and other stuff they REALLY wanted.

A major part of the meeting was devoted to talking about the upcoming CoCo Fest II. The date has been set for the 26-27th of June. The facility has been committed and the dinning arrangements have been selected. The event will be in Port Orchard at the HiJoy Bowl. Things will begin Friday evening with presentations until they throw us out. There is a NO HOST breakfast Saturday at about 8 am and then a luncheon at noon with a keynote speaker. We discussed the desirability of various novelty items for the FEST. Even though we had a mug last year, just about everyone wanted another mug. So we will have the second edition of the CoCo Fest mug. Also we will have a T-shirt. A great T-shirt, if I may say so. Discounts will be available for everything when ordered in advance. (See registration form elsewhere.)

Although the meeting was nothing like it was planned, goodness or success triumphed over chaos. We had a very worthwhile meeting.

But that's not the topper! When I got home I found out that Bob van der Poel, called me to accept the invitation to be the CoCo FEST keynote speaker. He will also attend the whole event so everyone will have a chance to talk with him on a 1:1 basis. Bob is the author of the famous VED Text Editor & Vprint Text Formatter. He offered also to talk at one of the workshops on the language C.

So this story has a more than happy ending. It has the potential of a fantastic convention for the CoCo community and a significant contribution ($160) to the Computer Bank Charity. What more could we want? "And so they lived happily ever after."

-- Donald Zimmerman --

NW CoCo Festival II

June 26th - 27th

Port Orchard, Washington