Introduction to UCSD Pascal System

By Ron Williams

A complete introduction to using the P-Code card on the TI-99/4A

Topics include:
P-System Editor
File Management
FunnelWeb
Reviews
Pascal Programming
Printing

Presented with detailed examples and plain language discussions

Published by
The Boston Computer Society
TI-99/4A User Group

Table of Contents

The Boston Computer Society's TI-99/4A User Group maintains a software library of over 100 disks. There are several UCSD Pascal disks available in this collection, including those mentioned in this booklet. Send $1 to the address below for a complete catalog.

If you wish to contact the author of this booklet, Ron Williams, please write care of the the address below.

The Boston Computer Society
TI-99/4A User Group
One Center Plaza
Boston, MA 02108

This material originally appeared as part of a monthly series in The Boston Computer Society's TI-99/4A User Group Meeting Newsletter. Many thanks to Ron for providing such an excellent column for so many months.

Editing and desktop publishing work to prepare for publication in booklet form by J. Peter Hoddie.

Introduction to the P-System

The UCSD P-System is a full operating system that supports the Pascal programming language and assembly language for the 99/4A. The UCSD P-Code card is required to operate this system as well as the UCSD Editor-Filer disk and the Compiler disk, also useful is a diskette with utilities that allow the user to do diskette initialization, device parameter modification as well as other useful functions. If assembly programming is to be done one more disk is needed the assembler/linker disk that allows the P-System programmer to assemble and link assembly programs to Pascal programs.

To start the system put the Editor-Filer disk in drive one and the compiler disk in drive two also if any other P-System disks are around put those in drive three as the system looks for P-System disks in the drives to boot properly. If the system is started without the disks in the drives do not panic as soon as the command prompt is displayed put the disks in the drives and press the "I" key for initialize and the system will boot again. At the command prompt many utilities are available like if the "F" key is pressed the system executes the Filer this program lets you do disk management functions and other system functions. If the "E" key is pressed this puts you in the editor used for creating assembly and Pascal text files also for creating documents. Other keys that do other functions from the main command prompt are as follows:

```
        "A" for Assemble
        "C" for Compile
        "H" for Halt (Returns to TI title screen)
        "L" for Link
        "I" to Initialize system
        "M" for Monitor (keeps track of keys pressed)
        "R" for Run (executes the last program compiled)
        "U" will restart last program executed
        "X" will prompt user for file to execute
```

Next time we will go over each of these functions and also how to set the system date, also how to read the P-Code disk directory.

Introduction to P-System Continued

Last month we begun to go over each of the functions of the main command menu, the commands are listed above.

The first command is the (A)ssemble command and this command starts the file System.assembler to execute, this will assemble the current work file if the system finds it on the disk in drive #1 or drive #4: as the P-system calls it. If the system cannot find the file it will prompt the user for a file to assemble. The file should be a text file created with the editor and it is a file that has 9900 assembly code. This code is not quite the same as the Editor-assembler code files that most TI people are familiar with because this is not the same assembler.

The next command is the (C)ompile command and this command also looks for the current workfile but this file is a UCSD Pascal text file. The compiler will try to compile this file into P-code this is a file the system can execute.

The (H)alt command is very simple as it is stated above it just returns you to the TI title screen.

The (L)ink command allows the programmer to link assembled code files and compiled UCSD Pascal files together as one program to pass data to and from each other much as the CALL LINK and CALL LOAD commands do in TI BASIC. The linker will also link two or more assembled code files together as well.

The (I)nitialize command causes the system to start just like it does when you first turn on the computer and the P-code card starts to take over the system. The system looks for the system files all over again and will execute the system.startup file if it is on the disk in drive one.

The (M)onitor command will keep track of keys pressed and will put these in a monitor file; this will let you automate a sequence of system commands to be used later. The file will be used by the system by redirecting the system input to the file.

The (R)un command will execute the file system.wrk.code in drive one. This is the file that is usually the work code file if this file is not found but a file called system.wrk.text is found the system will first compile this file then will execute the file. The Run command will always execute the current work file if one is assigned.

There is no prompt for the (U)ser restart command it will execute the last file executed but will not execute the files that start the compiler or the assembler. It is very good for executing one file many times as all files are held in memory until a new file overwrites them. This saves the system from getting the file from disk then having to execute it. The savings in speed make this a very useful command.

The last command is the "X" command, it will prompt the user for a file to execute. This command like the restart command will not execute the compiler or the assembler but will execute UCSD Pascal code file.

The last topic I would like to show you is how to set the system date. At the main command prompt press the "F" key with the filer disk in one of the drives when the filer prompt comes up press the "D" key and enter the date in the format shown on the screen.

Next time we will go into more detail about the filer as this is one of the most important programs in the system.

Filer and Edit

Last month we went over some of the commands that are started from the main command prompt, well there are two more that we have not discussed yet, they are the (F)iler command and the (E)dit commands.

This month as promised we will go over the (F)iler program. Make sure the filer disk is in one of the drives and press the (F) key when the main command prompt is shown this will put you in the filer program. The filer is menu driven and just like the main command prompt each command is started by a single key press. There are over 15 different commands and I will go over a few of the most important this month.

Some of the commands are as follows:

> "B" for Bad Blocks
> "E" for Extended Directory
> "L" for List Directory
> "C" for Change
> "K" for Krunch

The (B)ad Block command is used to scan the blocks on a disk to identify the blocks on the disk that are bad and cannot hold files. The blocks may then be repaired or marked bad with the examine command, but more on the examine command later.

The (E)xtended directory command lists the names of files on a disk as well as other disk information. The command will prompt you for the disk to catalog, and remember the disks are #4:, #5:, #9: and this catalog will default to the screen but may be directed to another device by putting a comma after the disk
name like #4:,#6: will redirect the output to the printer.

The (L)ist command is just a simple version of the extended directory command it is a little quicker than the extended directory command and may be used for a quick listing of files on a disk.

The (C)hange command is used to change the names of files or the disk volume name. The command will prompt you for the name of a file to change and remember to include the disk number with the file name or the change command will default to the disk in drive 1 or disk volume #4:. To change a file called MYFILE.TEXT on the drive #9: type in #9:MYFILE.TEXT and if the file is found the change command will then ask you for the new file name type in the new name like #9:CHANGE.TEXT and the file name will be changed to this new name.

The (K)runch command will consolidate the files together on the disk it is needed because the P-system has no way of fractioning files like the TI file system has. As files are changed on the P-system disks they are moved around to different places on the disk and so a large file may not fit on the small unused sections on the disk. This command will prompt you for the volume name to krunch then will ask you for the number of blocks to krunch and this number should be 180 for a TI single sided single density disk.

Well that's it for this month next month we will continue to look at the filer and I will show the use of wild cards.

Filer Continued

Last month we started to go over the commands of the filer program and this month we will continue, a few more commands are as follows:

"P" for (P)refix
"S" for (S)ave
"W" for (W)hat
"G" for (G)et
"M" for (M)ake
"T" for (T)ransfer
"R" for (R)emove

The (P)refix command lets you set the prefix to a volume so the disk can be referred to by a single colon ":". The root volume or drive #4: is automatically set as the prefix when the system is booted so this command can be used to change it to another disk.

The (S)ave command lets you rename the files SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE to other file names. These are the work files and this command lets you change these names to other names so you may create new work files.

The (W)hat command displays information on the status of the work file it will state what the file is, state that there is no work file, state the work file not saved or not named. This command may be used any time the current status of the work file is in doubt.

The (G)et command will assign a new file or pair of files as the current work file or work files. This command may be used any time a new file needs to be made the work file. It will assign two files without needing to know the suffix for the files. Both the text file and the code file will be assigned.

The (M)ake command will create a file that contains a certain amount of blocks. This command may be used to retrieve files that have been deleted or to reserve disk space for future use.

One of the most important parts of the filer is the (T)ransfer command it will be one of the most used commands in the filer. This command is used to make copies of files, display files on the screen or printer, reposition files and make backups of volumes.

The (R)emove command deletes files from the directory of a volume and marks the blocks as unused.

The use of the (R)emove command and the (T)ransfer commands may be simplified with the use of wild cards they may be used to match a sequence of characters in a file name. There are two wild cards in the system and they are the equals sign (=) and the question mark (?). The difference is the question mark will ask for verification of the command after the wild card is used, but the equals sign will not. To use a wild card to transfer a file like SYSTEM.WRK.TEXT to another disk volume type SYSTEM.WRK? after invoking the transfer command and then type #5:NEWFILE? and a copy of the file SYSTEM.WRK.TEXT will be on the volume in drive #5: as NEWFILE.TEXT also if there is a SYSTEM.WRK.CODE file on the root volume a prompt will be made if you would want to transfer this file too. If you transfer it, it will be called NEWFILE.CODE so the use of wild cards can save a lot of typing. The change command as well as the remove and transfer commands may be used with wild cards but it is advised the question mark wild card be used with the remove command.

Well that's it for this month, next month we will finish with the filer and start to look at the P-System editor.

Review of Persson Disk and Editor Continued

Before we start our lesson this month I would like to mention I have been reviewing the UCSD Pascal disks from Anders Persson of Sweden. Well, I am impressed as first on the disks is a map of the P-system memory addresses. Also included are a few units to be added to your system library or in a user library as I did. The units include a screen utility that will allow ACCEPT-VALIDATE as in Extended BASIC. Also routines that simulate CALL KEY and CALL GCHAR and as well as CALL VCHAR and CALL HCHAR. But the most useful to me will be the routine that allows you to scroll parts of the screen. Another unit that is included is one that allows you to create windows on the screen also a unit that will give a better response if you accidentally press (D)ebug at the command level. The last unit on the disks is one that allows you to use the mini-memory with the P-system to hold program data. On the disks are demo programs to demo these routines and a few other programs one that will allow P-system users to use four drives with the system but of course you must have a drive controller that supports this. One program that will also be of use to users of the P-system is an up date of the disk formatting program on the TI utilities disk with this program you may format double sided double density disks, the version on the utilities disk did not support this. Other programs that will be useful is a Disassembler, a Sector map program, and one that defines some key codes. These disks include doc's as well as source files for all the programs and units on the disks. These disks are a must for any user of the P-system.

This month we will finish up with our review of the filer and start with the P-System editor program. The last few filer commands are as follows:

"X" for e(X)amine
"N" for (N)ew
"V" for (V)olumes
"Z" for (Z)ero
"D" for (D)ate
"Q" for (Q)uit

The e(X)amine command is used to recover bad blocks on a P-system volume. If the blocks cannot be fixed with this command then you will be asked if you want to mark the blocks as bad, this way no files may be put in the bad area on the volume. Use this command after using the bad blocks command to repair the damage or mark the damaged areas bad.

The (N)ew command will delete the current work files and allow a new file to become the work file. Use this command after you have finished with the work files and saved them on another volume and would like to start a new work file.

The (V)olumes command gives you information on the volumes currently on line in the system. It also will show the root volume and the prefix volume. This command can be used to check volume's on line as some times I've had a volume go off line. With this command the name of a unknown volume can also be checked.

The (Z)ero command initializes a directory on a volume after a new disk is formatted this command must be used before you can store files on the disk. For a single sided single density disk the number of blocks on the disk should be 180 for the 99/4A, enter this number when zeroing a volume.

The (D)ate command sets the system date. The date is put on the root volume disk and is put on the disk next to any files updated in this session. The date command should be the first command used after booting the system.

The (Q)uit command is very simple, it just returns you to the main command prompt. Use this command when you have finished with the filer program.

The next program in the P-System I would like to look at is the editor. This is where you create your work files and can be used as a text editor for other applications. In fact this text was created with the P-System editor and then converted to a display variable 80 file to transfer. To start the editor press the "E" key at the main command prompt. If there is no work file you will be prompted for a file to edit. At this point enter a file name or press enter to start a new file. Before I go over the commands one point I would like to mention that most commands can be finished with control "C" and aborted with control period (.). The difference is that control "C" will keep the changes made with the command and control period (.) will cancel them. The other point is that the editor has some commands that in themselves have commands so I will go through each command and also follow up on each of their commands at the same time.

Well that's it for this month, next month I will start with the commands of the P-System editor.

Editor Continued

This month we will start with the commands of the P-system editor. The editor that the P-system uses is quite different from the other editor programs that the TI uses like TI-Writer. Each command is begun with a single key press like most of the P-system commands. This is different from TI Writer as when the cursor is displayed in TI-Writer you can add text and move the cursor around in the text. Well to insert text in the P-system editor you would press the "I" key and then you are in insert mode you may then start typing in the text. To go out of the insert mode you press control "C" then another command may be used. When out of insert mode the cursor may be moved around in the text with no danger of changing the text in the buffer you may press the space bar to go from one character to another or use the arrow keys to move the cursor through lines of text. To add text you may use the insert mode or another mode called the (X)change mode. This mode may be entered by pressing the "X" key and this mode will let you change one character at a time in the text it is normally used for small changes to a line of text but you can use the arrow keys in this mode to move the cursor around in the buffer if you need to.

Before I go on I would like to explain about the direction marker, the direction marker is the ">" character that is right at the top of the screen on the left it has a great effect on the use of certain commands like the page command if you press the "P" key when not in any editor mode the next "page" of text in the buffer will be displayed. The return key, the space bar and tab key are all effected by the direction marker. The direction marker is changed by pressing the ">" key or "<" key in edit mode or in delete mode.

To delete text in the buffer press the "D" key in edit mode(This is the mode you are in when first entering the editor) if you press the space bar one character at a time will be deleted in the direction of the direction marker. If you press return then all the text from the cursor to the end of the line will be deleted in the direction of the direction marker. Also arrow keys may be used in this mode, press control "C" to go back to edit mode.

Adjust mode will move the text the cursor is on in any direction by using the arrow keys use this mode to move a complete line of text left or right or to center the text. This mode is entered by pressing the "A" key while in edit mode.

The replace command is used to replace characters of text not unlike the Replace-String command in TI-Writer. This command is affected by the direction marker as the command will only look for the text from the cursor to the end of the buffer or from the cursor to the beginning of the buffer depending on the direction marker's direction. To use this command press the "R" key while in edit mode then put a delimiter up like the "/" character then the string to replace after this another delimiter and then another for the new string to replace the old string. To finish up put up one last delimiter and at this point the command will look for the string. The command can replace all target strings in the text buffer or just some of them by pressing the "/" key before the command is used or give it a number to replace.

The find command is very much like the replace command but it will only find a string in the text and put the cursor after it. It is also affected by the direction marker and can find all strings in the text or just the "n"th number. Enter this mode by pressing the "F" key while in edit mode. Both the find command and the replace command will find the last used target string if you press the "S" key after you go into the commands but with the replace command you still have to give it target string and a replace string if you press the "S" key twice this will use the last target string and the last replace string.

The quit command is the command used to exit the editor press the "Q" key while in edit mode and four prompts will be given, First (U)pdate press this key and the contents of the buffer will be written to a file called SYSTEM.WRK.TEXT on the root volume. Next is (E)xit it will return you to the main command prompt everything in the text buffer will be lost. The (R)eturn command will put you back in the editor at the same place you exited. The (W)rite command will put the contents of the buffer on any drive and you may give it any file name, this command is good when you have done a lot of changes to the text in the buffer and would like to write these changes out to a file and then continue using the editor. After this command writes out a file it will ask you to return to the editor or to go out of the editor program and back to main command mode.

The last command this month is the jump command pressing the "J" key will put you into this mode it will let you move quickly from the end or the beginning of the buffer by pressing "B" or "E" when in this mode it will also move from one marker in the text to another. A marker is an invisible character in the buffer the jump command and another command called the copy command use. If a marker is put in the text the jump command will put the cursor on the line the marker is on in the text. Markers are set with the marker option of the set command in the environment mode. Environment mode will be examined later as there is a lot to this mode.

Well that's it for this month, I think I have given you enough commands to at least play around with the editor some and find your way out of it again. Next month we will continue with the P-system editor.

Review of Coffey Disk and Editor Continued

Lately I have been reviewing a few disks put together by Jerry Coffey for the P-System. On the first disk is a way to create 40 or 80 track P-System boot disks with the Myarc quad density (80 Track) disk controller option. The disks are formatted DSDD in the 18 sector per track format. With this system more blocks are created on P-System disks to hold text, P-code, or data files. If you have drives and this disk controller, this disk maybe for you. The other disk that I have been looking at has a system that allows you to easily transfer P-System files over a modem using xmodem or te2 protocols. The program to do this is in three sections the first section takes P-System files (Text or Code) and converts them to display/fixed 128 files that can be easily transferred by many terminal programs the TI supports, then the programs can be converted back to regular P-System files and read, executed or printed from the P-System. Also in this program is a section to convert the display/fixed 128 files to display/variable 80 files that can be read by TI writer or other text editor programs. The program also contains a section that will partition a disk into two sections. One section is in regular TI format that most programs can handle, the other section is a zeroed P-System volume and this section can be any size the user wants. Also on this disk is a valid P-System volume as this disk was partitioned using the program on this disk. This volume contains a few programs for the P-System and they include Andy Cooper's P-System terminal emulator, a disk cloner for many different size P-System disks, a character set by Dave Ramsey, a program that will bring display/variable 80 files into the P-System, a file printing program, a program that will set the Myarc disk controller to multiple format, a program that will automatically set the printer to PIO and RS232/1 at 1200 baud (This will be needed for Andy Cooper's terminal program), a program that will automatically set the printer to PIO at 300 baud and lastly a program that will read a memory range set by the user. Both these disks include documentation and may be already well known by many P-System users but I thought I would pass this along anyway.

Now to continue with the P-System editor program, as we started with last month. The copy command has two parts if you press "C" while in edit mode you are prompted to copy from the buffer or a file. If you copy from the buffer all contents of the copy buffer will be moved to the screen and inserted in the text. The copy buffer is a buffer that holds text that has been most recently deleted or inserted and using this command it is possible to duplicate text anywhere in the main text you are working with. This can save a lot of typing. The file option of the copy command lets you take text from a file outside of the editor program and insert this text into the text you are working with and if markers are set in this text file you may copy from marker to marker in this file.

The verify command displays a portion of the text buffer that is centered around the cursor and will let you display a portion of the screen maybe in between pages of text. Use this command also if you have a question of what the text buffer really contains.

The zap command lets you delete large portions of text. The text deleted will be the text from the cursor to the text last inserted, found or replaced and this command will put the deleted text in the copy buffer so it can be copied back out to another place. If the text is too large for the copy buffer you will be prompted to this, so use this command carefully.

The command that is most complex in the editor is the environment option of the set command, this command sets the editor to different modes much like the word wrap mode and fixed modes of TI writer. All changes are set from one screen and also displayed is the number of used bytes and unused bytes, the targets used by the find and replace commands, the current marker names and the date the file was created and the most recent update. The following is a list of the commands and a brief description of what they do:

Auto Indent	: Controls where the cursor will be after return is pressed.
Filling	: Controls the filling of words in a line to the right margin.
Right Margin	: Sets the value of the right margin.

Para Margin	: Sets the indentation for the first line of a paragraph.
Command Ch	: Currently not used.
Token Def	: Sets the default search mode for the find and replace commands.

The set command when it is first displayed will also allow you to set markers in your text. Position your cursor where you want the marker, press the "S" key for set and go to marker mode enter the name you want the marker to have and it will be put where the cursor is set. Press the space bar to return to edit mode after changes are made with the set command.

The last command that is closely tied to the set command is the margin command. The margin command lets you format a paragraph differently from other paragraphs in the text and will format the text according to the margin settings in the set command (Environment option). The margin command can also reformat a paragraph after a delete in text mode and reformat paragraphs that were inserted while the editor was not in text mode.

Next month we will start with a series on execution options with the execute command and also on adding units to the P-System library.

Execute and Library

This month we will do a section on execution options with the execute command. The execute command is entered from the main command prompt. Normally you would just type the name of a volume or volume number followed by a colon and the file name of the file to execute the computer will then look on the drive for the file name and execute the file. This is good for normal use of most programs but what if you wanted the program to look to a file for input data or if you wanted the program not to place its output to the screen but say to the printer?. Well this is where the use of execution options come in. With them you can redirect program output as well as system input. Most of the options are used by placing the option after the filename of the program to execute. But a few may be used without any file to be executed. One of these is the "P" option this option will change the default volume prefix like the prefix command in the filer program will do. To use this option press the "X" key from the main command prompt just like you where going to execute a file and instead of putting up a filename put up on the screen "P=" and the new volume name for a default prefix. The same can be done for the library textfile press "X" again and this time put up "L=" and the new library textfile. All programs now executed that look for a library text file will look for this file to find their library. Normally most programs look to the screen for input but this may be changed also by using the option "PI". If a program is to be executed after you place the program with volume name up on the screen put "PI=" and a filename all input for the executed program will come from this file. To redirect the program output just use "PO=" instead of "PI=" all output from the program will go to this file. Like if you wanted program output to go to the printer put up "PO=PRINTER:" after the file to execute. To redirect system output two options are available "I" for system input and "O" for system output the system will redirect its output to any files after this option. Using redirection can be a powerful tool as programs that need to be tested may redirect their output to a file to be saved and looked at later. How many times when debugging a program in Extended BASIC have you wished you could have saved the output to study later, well this is no longer a problem with the P-system.

The next topic I would like to look at is adding units to a library file. The program LIBRARY.CODE on the UCSD utilities disk is the program that lets you do this. Go to the main command prompt and press "X" to execute the file, adding the file "LIBRARY" after the prompt "what file?". The first prompt the program will give you is "output file?" this is the file that the units will be placed after you are finished with the library program. Enter a file name and press enter. After this prompt "input file?" appears type the name of a file to be used as the input file. This file may already have units in it if so their names will be placed on the screen. The units in the input file must be placed in the output file before any new units can be added so you can press the "E" key to copy every unit to the output file or type a slot number and then press the space bar and another number of a slot in the output file and the unit will be placed here, or another way to transfer them is to press the "S" key to select the units to transfer to the output file you will then be prompted to transfer each unit. Enter "Y" or "N" to transfer each unit. Most of the time you will use the "E" key to transfer all units to the output file but may be you want to add an updated unit to the output file then you must transfer some of them but not all of the units to add the updated unit. Not only units may be put in a library file but programs, segment routines and assembled routines may be put here as well to interface with programs. To add a new unit enter "N" and a filename of the new unit this unit will be put in slot 0 of the input file transfer this file to the output file in one of the remaining slots and it is now in the new library file. Press the "Q" key to exit the library program and save the library file. Other features of the library program are:

Press the "C" key to copy to the output file a compilation unit.

Press the "F" key to copy all segments referenced in the output file from the input file.

Press "O" to display the remainder of the output file list.

Press "A" to abort the program without saving the library output file.

Press "I" to display the remainder of the input file list.

Press "R" to list the names of each entry in the segment references of all segments in the output file.

Press "T" to determine whether or not the interface sections of the units are copied to the output file.

The use of units can add a lot to your programs as some programs can use the same units as other programs just by adding the "USES" statement in your programs. One example of this is a unit I created that gets the system date from the system disk in drive #4 many programs I use create reports or need the date to be printed out. This way I don't need to type in the date each time I run the programs. I add the date to the system after booting and that is all. All these programs just call the unit to get the system date each time they need it.

Well that's it for this month, next month I'm not sure what I'll cover, may be something on the Pascal language we'll see.

P-System and FunnelWeb

This month I will cover a different subject pertaining to the P-system. First any one using the P-system has noticed if they exit the P-system environment and run assembly programs many times after exiting the program the P-system will boot again and this is not always wanted. Well if you use FunnelWebas I do, there are ways around this rebooting. I try to keep within FunnelWeb as much as possible as long as a program is running the P-system will not boot but it will boot if you exit to the title screen.

To exit FunnelWeband go back to the title screen without the P-system rebooting you will need to assemble the following assembly code:

```
        DEF PCODE
        REF VMBW
NO      EQU >4E4F
DAT1    TEXT 'P-CODE HALT BOOT PROGRAM'
DAT2    TEXT '(C)1987 RON WILLIAMS     '
PCODE   LI R3,NO
        MOV R3,@>38FA
        LI R0,1
        LI R1,DAT1
        LI R2,24
        BLWP @VMBW
        LI R0,33
        LI R1,DAT2
        BLWP @VMBW
        RT
        END PCODE
```

This program will move the proper ASCII codes to the address the P-code card looks at to see if it should boot or not. If it finds the word "NO" ASCII codes in hex >4E4F the system will not boot and will exit to the title screen. The opposite is also true to make the P-system boot just put hex >0000 at this address and the P-system will start instead of exiting to the title screen. You may have noticed that in the assembly code it returns you to FunnelWeb and not direct to the title screen well this program has another use some assembly programs run from FunnelWeb, if this program is run just before executing them they will exit to the title screen after they are finished executing. This way you can keep working with programs outside of the P-system with not going through a P-system reboot. This program will not keep all programs from rebooting the P-system but after a while you will know the programs it will work with and the programs it will not work with. I use this stop boot program just before exiting FunnelWeb or executing another assembly program from funnelweb. When ready to exit FunnelWeb run this program and press FCTN QUIT instead of pressing the exit key on the funnelweb menu screen if you press the exit key the P-system will boot so that is why I press FCTN QUIT. I have put this program on my user list in FunnelWeb and that way it is ready for use anytime I exit FunnelWeb, the program is very small and doesn't take much room on a disk.

To stop the P-system from booting while in Extended BASIC before typing BYE or pressing FCTN QUIT you can type:

```
CALL INIT
CALL LOAD(14586,78,79)
```

The number 14586 is the address >38FA in decimal and 78 and 79 are the ASCII codes for "NO".

To make the P-system boot while in extended basic before typing BYE or pressing FCTN QUIT you can type:

```
CALL INIT
CALL LOAD(14586,0,0)
```

To check the proper values are loaded you may type:

```
CALL PEEK(14586,A,B)
PRINT A
PRINT B
```

where A and B are the values checked.

Edit, Compile, and Execute

This month I will cover the complete number of steps needed to edit, compile and execute a Pascal program. First boot up the P-system by putting the P-system disks in each drive and turning on the computer. The first thing that should be done after booting is to press the "F" key to go into the filer and then press the "D" key to set the system date. This way all files and programs created will have the correct date next to them when doing a directory of the disk. The other reason that you would want the date changed is any updates to a file or program will have a new date, any program updates can be checked by the date created. The way to get out of the filer program after setting the correct date is to press the "Q" key and this will put you at the main command menu. To start the editor press the "E" key and you will see a prompt "no workfile present, file?" press enter at this point or if you want to load a file into the editor enter its name by typing its drive number like this #4: followed by the file name like this #4:myfile the .text suffix will automatically be appended to it. For this example we will assume there is no file present so press enter and you will see "edit:" followed by a few commands the cursor will be in the upper left hand side of the screen waiting for a command. To start entering text press the "I" key to insert text. Enter the following text as an example:

```
Program count;
```

When you get to the semicolon press enter and the cursor will drop down to the next line if you make a mistake press fctn arrow to go over the error then retype it. If you make a mistake and notice it after you press enter press control-c and this will take you out of insert mode and at this point all arrow keys now work, press fctn arrow to put the cursor over the mistake and at this point you may press the "X" key to type over an incorrect character, or use the insert mode to insert text on this line after finished with either mode press control-c to go back to command mode.
Type the rest of this text as follows:

```
Program count;
  var
    number : integer;
  begin
    page(output);
    for number:=1 to 10 do
      writeln(number);
    page(output);
  end.
```

This program will print out the numbers from 1 to 10 and is very simple but it will be good enough as an example program. If you need to delete a line of text or just one letter put the cursor over it with fctn arrow keys while in command mode then press the "D" key to delete one letter press the space bar or to delete a line just press enter to go out of this mode press control-c. The adjust mode is entered by pressing the "A" key all arrow keys now work. To move over a line of text put the cursor on it and use the arrow left and arrow right keys to move it. After the program is entered press "Q" while in command mode. The next screen you will see has the following prompts:

```
>QUIT:

U(pdate the workfile and leave
E(dit without updating
R(eturn to the editor without updating
W(rite to a file name and return
```

Press the "U" key to write out the file in the editor's buffer to drive #4: with a file name of system.wrk.text this is the default name used when updating. Exit will return you to command mode and all contents of the buffer will be lost. Return will just put you back in the editor use this key if you pressed "Q" by mistake. Write will let you save the buffer to any file name or drive use this key when you wish to save out a file and then return to the editor. The file will be written out to disk then you will be given a chance to exit or return to the editor.

After editing the program you must now compile it and execute it. To compile a program saved as system.wrk.text press the "C" key at the main command menu or press the "R" key to compile and execute the program all in one step. When compiling the screen will first go blank then at the top of the screen you will see compiling... then the version number.
The compiling process looks like this:

```
Pascal compiler - release 99/4 IV.0 cla-4
<0    >....
COUNT
<4    >....
8 lines compiled

count   .
```

If any errors are found you can directly go back to the editor or continue the compiling process. If the "R" option was used the program will begin right after compiling if not you can execute the program from the main command line by pressing "R" or pressing the "X" key and entering the file name of the program which will be for this example system.wrk you don't need to add the .code suffix when executing a program as it will be added automatically. If you did add it put a period after the "e" in code this will make the computer find the correct file and not system.wrk.code.code.

Well that's it for this month I hope this was useful to you. I have had a few people ask me about this in the past. So long until next month.

Introduction to Pascal

This month I will begin to cover some Pascal programming. Pascal is a language that like "C" is compiled, you must first type in the program with a text editor and then compile the text file to create an executable code file. The UCSD Pascal that the TI supports is the same UCSD Pascal that you could get for an Apple, IBM or other microcomputer the basic difference is memory, the TI just does not have as much but with some special programming you could get a lot of the Pascal programs written in UCSD Pascal on other machines to run on the TI. I have taken code that was created on other computers and with a few changes compiled and ran them on the TI. This code was brought in as text files and not code files, you still have to compile the files on the TI, as code files on other machines is compiled in that machine's assembly code.

The very first statement in every Pascal program is the program statement and it is at the beginning of every Pascal program.

It looks like this:

```
program Hello;
```

This statement must be included at the beginning of every Pascal program and it is changed for every program written, it should have a name of what the program is going to do. The declaration statements are next and there are three of them Constant, Type, and Variable and they are used in the order given but not all of them are used at any one time. The Constant declaration is used to declare program constants, these are identifiers that are never changed thru the execution of the program.

A example of a constant declaration is as follows:

```
Const
  pie = 3.14;
```

The next declaration statement is the type declaration and it is used to make a user defined type, that is you may create a record, or another type that is used by the program to mean something.

An example of the type declaration is as follows:

```
Type
  Tens = (10,20,30,40,50,60,70,80,90);
```

This means that the identifier tens can have the value of the numbers that follow it and no others, it is a type defined by the programmer. The last declaration statement is the variable statement. It is used to declare program variables, these identifiers can be changed by the program and all identifiers must be declared before using them unlike BASIC which you can make a new variable any time you need it. This means it is very easy to know what variables are being used by the program and you could also put comments next to them to tell anyone reading the program what they are.

An example of the variable statement is as follows:

```
Var
  Number : integer;
```

When you declare an identifier you may use some data types that are already defined for you within the Pascal language and a few of them are:

Integer	This type is for numbers that are to be defined as integer

	types(numbers without decimal points).
Real	This type is for real numbers(Numbers followed by a decimal point).
String	This type is used for characters of data.
Char	This type for one byte characters.
Boolean	This type accepts only a value of true or false.

Next month we will look at a few statements that the Pascal language uses to print out data and read data and also at the begin and end statements.

Writing and Looping

Last month I begun to cover Pascal programming this month we will look at the write statement, the read statement, begin and end statements, and also at loops. The write statement allows you to print data to the screen, the printer, or a Pascal text file. It will print character data or numeric data.

An example is as follows:

```
WRITE('hello');
```

This statement will print out hello and will cause any other data printed out to be on the same line as hello.

The statement

```
WRITELN('hello');
```

Will let any other data printed to be on the next line the writeln statement will make a carriage return after it. Now to get the data to be printed on other columns you could write the statement like this:

```
WRITELN('hello':10);
```

This will make the data to be printed on column 10 right justified. Real numbers may also be printed, formatting them also to print out the number of decimal places to the right of the decimal point.

Like this:

```
NUMBER:=3.14;
WRITELN(NUMBER:5:2);
```

The second number after the five is the number of decimal places to be printed.

The read statement will let you input data from the screen or a Pascal text file and is like this:

```
READLN(INDATA);
```

With INDATA being any variable previously defined. The read statement will cause any data printed out to be on the same line as the read input data so use readln if you want the next data read or printed to be on a different line. The read statement will read characters of data, a character, or real or integer numbers.

The begin and end statements are very important as they define groups of statements to be used by the program. To begin a program the main body must have a begin and end statement and the last end statement should have a period after it. A lot of statements in Pascal also need the begin and end statements to tell where they begin and where they end. A good example of this is the for loop, this loop is very much the same loop used in BASIC.

An example is as follows:

```
FOR X:=1 TO 10 DO
  BEGIN
     WRITE('NUMBER');
     WRITELN(X);
  END;
```

This example uses the write statement, and begin and end statements tell the beginning and the end of the loop. The for loop does not really need the begin and end statements all the time if there is only one statement after it the semicolon after the statement will be the end of the loop.

Like this:

```
FOR X:=1 TO 10 DO
  WRITELN('NUMBER',X);
```

The next loop to show you is the repeat loop it is like this:

```
REPEAT
  <PROGRAM STATEMENTS>
    .
    .
    .
UNTIL CONDITION;
```

This loop uses a condition defined after the until to exit the loop, This is a boolean condition telling the loop to end.

The while loop is like the repeat loop but the condition is checked before entering the loop.

Like this:

```
WHILE CONDITION DO
  BEGIN
    <PROGRAM STATEMENTS>
      .
      .
      .
  END;
```

The begin and end statements are required for this loop and it is good loop for checking data before entering a loop.

One more thing I wanted to mention is the use of semicolons you may have noticed these after some of my program statements. These are required in the Pascal language after a lot of Pascal statements like the write and read statements and after the end statement in a group of statements.

So long until next month.

P-System and Funnelweb Continued and Accessing Text Files

A few months ago I had a article about how to stop the P-System from booting when exiting Funnelweb well I have made some changes to the source code and came up with the following program.

The complete source code is as follows:

```
*****************************************************
* THIS PROGRAM WILL LET THE FUNNELWEB USER BOOT     *
* THE P-CODE CARD OR WILL LET THE USER STOP THE     *
* AUTO-BOOT AND RETURN BACK TO THE TITLE SCREEN     *
* THIS PROGRAM MAY BE ADDED TO ANY FUNNELWEB MENU   *
*****************************************************
        DEF BOOT
        REF VMBW,VSBW,KSCAN
NOBOOT  EQU >4E4F
YEBOOT  EQU >0000
RETURN  BSS 2
WR      BSS >20
LINE1   TEXT '1. STOP BOOT'
LINE2   TEXT '2. BOOT P-CODE'
BOOT    MOV R11,@RETURN        MOVE RETURN ADDRESS TO HOLDING AREA
        LWPI WR                LOAD WORK SPACE
        CLR R0                 CLEAR REGISTER
        LI R1,>2000            LOAD SPACE CHAR TO REGISTER 1
CLSR    BLWP @VSBW             EXECUTE FUNCTION
        INC R0                 INCREMENT REGISTER
        CI R0,768              COMPARE
        JLT CLSR               JUMP IF LESS THAN
        LI R0,362              LOAD SCREEN POSITION
        LI R1,LINE1            LOAD DATA
        LI R2,12               LOAD LENGTH
        BLWP @VMBW             EXECUTE FUNCTION
        LI R0,426              LOAD SCREEN POSITION
        LI R1,LINE2            LOAD DATA
        LI R2,14               LOAD LENGTH
        BLWP @VMBW             EXECUTE FUNCTION
LOOP    LI R2,>2000            LOAD SPACE CHAR TO REGISTER 2
        BLWP @KSCAN            EXECUTE READ KEY BOARD
        MOVB @>837C,R1         MOVE STATUS TO REGISTER 1
        COC R2,R1              COMPARE
        JNE LOOP               JUMP IF NOT EQUAL
        MOV @>8374,R1          MOVE KEY PRESSED TO REGISTER 1
        SWPB R1                SWAP BYTES IN REGISTER 1
        CI R1,>3100            COMPARE TO ASCII KEY "1"
        JEQ SBOOT              JUMP IF EQUAL
        CI R1,>3200            COMPARE TO ASCII KEY "2"
        JEQ YBOOT              JUMP IF EQUAL
        JMP LOOP               KEEP IN LOOP UNTIL KEY IS PRESSED
SBOOT   LI R3,NOBOOT           MOVE NO BOOT CODE TO REGISTER 3
        MOV R3,@>38FA          MOVE VALUE TO MEMORY LOCATION
        JMP ENDPRO             JUMP TO END PROGRAM
YBOOT   LI R3,YEBOOT           MOVE BOOT CODE TO REGISTER 3
        MOV R3,@>38FA          MOVE VALUE TO MEMORY LOCATION
ENDPRO  CLR @>837C             CLEAR STATUS
        MOV @RETURN,R11        MOVE RETURN
        BLWP @0                BRANCH TO TITLE SCREEN
        END
```

The original program may still have some uses like I stated in the previous article some programs may be run after executing the original program and the P-system may not boot, this program just gives you two choices to boot the system or to return to the title screen and no boot. I currently use this program when I am in Funnelweb and want to return to the menu of my ramdisk but it will take you to the main title screen if you do not have a ramdisk or the ramdisk is not set for auto-start.

I will now continue with more Pascal programming using what we have covered so far we should be able to write some very simple programs. So adding a few new statements I will show you how to read Pascal text files. I will set up the program first with comments to explain what is going on.

The program is as follows:

```
program readtext;

(* This program reads a *)
(* Pascal text file the *)
(* file name is input   *)
(* by the user          *)

  var
(* define input string *)
    line : string;
(* define file type *)
    textin : text;
(* define string for file input *)
    fname  : string;
  begin
    page(output);              (* clear screen *)
    write('enter file name=>');
    readln(fname);             (* read file name *)
    reset(textin,fname);       (* open file *)
    while not(eof(textin)) do  (* stop input at *)
      begin                    (* end of file   *)
        readln(textin,line);   (* read in one line *)
        writeln(line);         (* write out line to *)
      end;                     (* screen            *)
    close(textin,lock);        (* close file        *)
  end.
```

The program uses a few of the statements that we have gone over before but it also shows other commands. To clear the screen, this command is just about the same as the CALL CLEAR command in TI Basic, the Reset command is very close to the OPEN file command in Basic except that the file type has already been defined and also there is no file number assigned. The drive to get the file is added to the file name when you input it. To get a file from drive number 5 which is DSK2 input the file like this #5:THEFILE.TEXT. Remember that this program reads text files (Any file with the .TEXT Suffix) if you try to read a file that is not a text file all it will do is really mess up your screen display. The while loop will just read the file until end of file (eof) and write it out to the screen. After this the file is closed and locked this statement can also delete the file by putting PURGE in place of LOCK and other options are available. The program is now ended with the final statement END. the period at the end tells the compiler that this is the end of the program and not just the end of another procedure or function within the program.

Case and If-Then

This Month I will show you a few new statements of the Pascal programming language. The first statement is the Case statement, it is very close to the On-Goto or On-Gosub statement in Basic.

The Case statement is shown below:

```
CASE number of
   1 : write('one');
   2 : write('two');
   3 : write('three');
   4 : write('four');
   5 : write('five');
   6 : write('six');
   7 : write('seven');
   8 : write('eight');
   9 : write('nine');
  10 : write('ten');
end;
```

The identifier number is declared as as an integer type number. If the value of the identifier is any number 1-10 the Case statement will cause the statement following the Colon : to be executed for example if the value of number was 3 the program will print out the string "three". In UCSD Pascal this statement is ok to be used in a program but in standard Pascal and possibly other Pascal languages this statement may not work if the value of the number is another value and not 1-10 but in UCSD Pascal if the value is not found the statement will simply not print out anything and there will be no errors. This could become a problem if you are trying to convert this statement to be used with another Pascal language.

The If statement is close to the same statement in Basic but it is much simpler
to use.

An example is as follows:

```
If number = 1
  then
     write('one')
  else
     write('number not equal to one');
```

This example shows that the IF statement can also include an else clause if needed also the statement may use begin and end statements if a condition is met and there are a lot of things to be done. The If statement can also call a Procedure or a Function within a program but I will go into more detail about Procedures and Functions later. The If statement can also be a lot more complex than this simple example, other boolean operators may be used like OR, AND, NOT to make this statement one of the most useful in Pascal the statement may look like this:

```
If (number = 1) or (number = 3)
  then
    writeln('stop');
```

The If statement will be used a lot but keep in mind that in some cases the CASE statement may work a lot better like in the first example I showed you if you were to write this using If statements it would be very long and drawn out so use the statement that will work with the least effort. Well so long until next month.

Using String Data Types

Many of the same functions you can use in Extended BASIC are available for use in Pascal and are just as convenient to use as in Extended BASIC. The string type is used a lot in UCSD Pascal and can be defined in the type section or in the variable section. When you declare a string you can define the string like this:

```
VAR
  word : string[10];
```

This string has a max length of 10 characters and if you try to give the variable a longer length it will chop off the extra characters on the end. The string type also has a default length and that is 80 characters and a string can have a length as short as 1 character and a max length of 255. The procedure readln should be used to import strings in to a program but it may be necessary to make a string another way because using string types is much easier than using a packed array of characters. One other way to make a string is to convert a packed array of characters to a string and that is done like this:

```
word:='          ';
for count:=1 to 10 do
   word[count]:=multichar[count];
```

The variable word is of type string and the variable multichar is a packed array of characters. You must give word a length before assigning the variable one character at a time or you will get an execution error that is why I put the assignment statement before the loop. Another way to give word a new length would be to use a compiler directive to shut off range checking and then assign the new length directly like this:

```
(*$R-*)
word[0]:=chr(10);
(*$R^*)
for count:=1 to 10 do
  word[count]:=multichar[count];
```

This method may be used if you are not sure of the new length of the string and how long to set the loop for. The chr function can except a variable as well as the constant "10" for input.

Some string procedures and functions included in UCSD Pascal are explained below:

The function concat will put together a number of strings and each string must be put in the function separated by commas like this:

```
        CONCAT('hello',' how',' are you');
```

It will return: hello how are you

The function copy returns a string of characters starting with a position and a size the function could be used like this:

```
        COPY('hello how are you',7,3);
```

It will return: how

The function pos returns a integer value it returns the starting position of a string within a source string. It is used like this:

```
        POS('are','hello how are you');
```

It will return: 11. This is the first place in the string that 'are' is found.

The function length will return a length of the string it returns an integer value it is used like this:

```
        LENGTH('hello how are you');
```
It will return 17 , an integer.

The procedure delete will remove characters from a string it will remove the characters starting with position and ending with a size. The procedure is used like this:
```
        line:='hello how are you';
        DELETE(line,1,6);
        WRITE(line);    (* will return 'how are you' *)
```
It will remove 'hello' from the string.

The procedure insert does just the opposite as delete it will insert a string into a source string it is used like this:
```
        line:='hello how are you';
        INSERT(line,',hello',6);
        WRITE(line)     (* will return 'hello,hello how are you' *)
```
It will insert the string ',hello' into the variable "line".

The next string functions that I will show you are not found as part of UCSD Pascal but have been added to the user library of the TI 99/4A the additional functions are in the library misc so put uses misc as a library at the beginning of your program.

The function break will return the position of the first character in the source string that matches a character in the second string it will be used like this:
```
        thepos:=BREAK('hello','0');
```
The variable thepos will now have the value 5 the string 'o' is the 5th character in 'hello'.

The function span will return the first position of a character not found in the source sting it is used like this:
```
        thepos:=SPAN('hello','h');
```
The value of 2 will be put into the variable thepos because 'e' is the first character not found.

The function upper-case will return upper-case letters in a new string with the source string having lower-case letters.
```
        line:='hello how are you';
        UPPER_CASE(line,line2);
        WRITE(line2);   (* will return 'HOW ARE YOU' *)
```
The function will convert the lower case letters in line to upper case and put the string in line2 the source string is not changed and so a new string is created with upper case letters.

One more point I would like to make, is make sure you use READLN and not READ with strings the results when using READ can be very bad, things like run time errors and other problems can develop. Well that's it for this month, thanks.

Formatted Output

This month I will cover the writeln procedure in more detail. This procedure is one of the most used procedures in Pascal as most of the writing to the screen and other devices will use this procedure. The writeln procedure writes text files and can not write records like the procedure PUT can. One of the great things about writeln is that it can also format output to the screen, printer or other device when writing out the data.

The following program is a demo of formatting with the writeln statement as you can see I have gave you the option of directing the output to the screen or to the printer. If you output to the printer you can study the output in much greater detail.

```
program testwrite(input,output);
const
  pi = 3.14159;
var
  pfile : text;
  choice : char;
  count,count2 : integer;

(* This program will demo the *)
(* use of writeln in Pascal   *)
(* The program uses loops to  *)
(* show output formatting     *)

begin
  page(output);
  gotoxy(1,1);
  write('(1)screen  (2)printer=>');
  read(choice);
  case choice of
    '1'  : rewrite(pfile,'console:');
    '2'  : rewrite(pfile,'printer:');
  end;
  gotoxy(1,4);
  for count:=1 to 6 do
    writeln(pfile,pi:8:count);
  writeln(pfile);
  for count:=1 to 8 do
    writeln(pfile,pi:count:1);
  writeln(pfile);
  for count:=1 to 8 do
    for count2:=1 to 6 do
        writeln(pfile,pi:count:count2);
  close(pfile,lock);
  page(output);
end.
```

This program will first show in the first loop how to print a real number with a different number of spaces after the decimal point. The first value after the colon : is the number of spaces to print over from the left edge of the paper or screen and the second number is the number of decimals to print after the decimal point notice as the values in the loops change the output also changes. If the valve given is not really possible like printing one space over after wanting two spaces after the decimal point the program will print the number in scientific notation so make sure the values work.

A sample of the output from the first loop is shown below:

```
     3.1
    3.14
   3.142
  3.1416
 3.14159
 +3.1415900000000e+000
```

A sample of the output from the second loop is shown below:

```
+3.1415900000000e+000
+3.1415900000000e+000
+3.1415900000000e+000
3.1
 3.1
  3.1
   3.1
    3.1
```

So far I have shown output with real numbers well writeln can also output strings, integers, and packed array of characters. The only real difference is that you would not use two colons like with real numbers. The number after the colon would only be for the number of spaces to print over from the left edge of the paper or from the last write on the same line. The value of the number after the colon if not possible will try to print the number or string as well as possible and if to small will chop off any characters on the end. One more point I would like to make is that write and writeln is right justified unlike print in BASIC I know this will sometimes be a problem for beginning Pascal programmers but soon you will grow to get used to it and may even find it much easier to format the printout of numbers I know I did. Just keep in mind that the number will be printed from the left over X amount of characters. Well that's it for this month, Thanks.

Printing

This month I will try to do something a little different I will go into setting your printer for different type print using UCSD Pascal I think you will find it to be quite easy. I will also give you a small printer setting program or procedure that you could use as a program all by itself or as a procedure you could add to a program. I use a very similar procedure in a few programs I currently use to set my printer. I have found that many times programmers in the P-system do not put in printer setting procedures in their programs this could be because there are so many different printers being used well I will show you how all the commands work so if my program will not work with your printer you could change the program to make it work with your printer.

The program is as follows:

```
Program print;

USES SUPPORT;

 var
   pfile :text;
   answer : char;
   code : integer;

 procedure codes;
   begin
     gotoxy(1,21);
     write('Enter codes one number at a time');
     repeat
       gotoxy(1,22);
       write('Enter 500 when finished=>');
       readln(code);
       gotoxy(26,22);
       write('       ');
       if code <> 500
         then
           write(pfile,chr(code));
     until code = 500;
     writeln(pfile,chr(7));
     gotoxy(1,21);
     writeln('                                ');
     writeln('                                ');
   end;

begin
  rewrite(pfile,'PRINTER:');
  page(output);
  set_screen(0);
  set_scr_color(1,8);
  set_screen(1);
  gotoxy(1,1);
  writeln('printer set up');
  gotoxy(1,3);
  writeln('(1)condensed pica');
  gotoxy(1,5);
  writeln('(2)condensed elite');
  gotoxy(1,7);
  writeln('(3)skip perforation');
  gotoxy(1,9);
```

```
    writeln('(4)double strike');
    gotoxy(1,11);
    writeln('(5)letter quality');
    gotoxy(1,13);
    writeln('(6)expanded print');
    gotoxy(1,15);
    writeln('(7)cancel paper out');
    gotoxy(1,17);
    writeln('(8)reset printer');
    gotoxy(1,19);
    writeln('(9)enter your codes');
    repeat
      repeat
        gotoxy(1,21);
        write('enter choice=>');
        gotoxy(15,21);
        readln(answer);
        case answer of
         '1' : writeln(pfile,chr(15),chr(27),chr(71),chr(7));
         '2' : writeln(pfile,chr(27),chr(77),chr(15),chr(27),chr(71),
                         chr(7));
         '3' : writeln(pfile,chr(27),chr(78),chr(6),chr(7));
         '4' : writeln(pfile,chr(27),chr(71),chr(7));
         '5' : writeln(pfile,chr(27),chr(120),chr(49),chr(7));
         '6' : writeln(pfile,chr(27),chr(87),chr(1),chr(7));
         '7' : writeln(pfile,chr(27),chr(56),chr(7));
         '8' : writeln(pfile,chr(27),chr(64),chr(7));
         '9' : codes;
        end;
      until answer in ['1'..'9'];
      gotoxy(1,21);
      write('continue set up Y/N=>');
      readln(answer);
      gotoxy(1,21);
      write('                          ');
   until (answer = 'N') or (answer = 'n');
   close(pfile,lock);
   page(output);
 end.(*print*)
```

This program should work with most Epson compatible printers I think the worst that could happen that the one touch key presses would not work the enter your own codes section should work with just about any printer. This program shows you just how easy it is to change your printer settings in UCSD Pascal. The program starts with a uses statement this is to get a function out of the library this function will let me set the screen color you can change it to anything you want the first number in the function set_scr_color is the color of the text and the second number sets the screen around the text. The program currently sets the text to black and the screen to medium red. After this the program will display the nine choices some you can set the printer with one key press but the ninth choice will let you enter you own codes this is because there are many printer setting codes and I could not put all of them in the menu of choices with out the menu being very large. The ones I did choose are the codes I use most of the time. Try going to the enter your own code section and setting the printer to condensed elite print with subscripts some very small text is possible. My codes are 27,83,1 for subscripts and then press condensed elite after typing in "500" to exit this procedure the text printed out on the printer will be very small you could also change the line spacing to make the print more spaced evenly. I sometimes like to print disk directories like this for disk jackets and now you can do this in the P-system by printing out a volume directory in the Filer program after setting the printer. Remember that the printer will be set this way even after you exit the P-system shutting off the printer or printer reset is the only way to set your printer back the way it was. The program uses the chr function quite a bit this is very close to the chr$ function in Extended BASIC. The chr function will let you set the printer codes, by putting in the proper codes from your printer manual you can set your printer for anything it is capable of doing. Also you can put in your own codes as needed if some of these codes do not work with your printer. Try entering in the codes first using the enter your codes section of this program to make sure they work correctly. This program also uses the case statement which is somewhat like the on goto function in BASIC. This function lets each statement to set your printer work separately. Well I hope this program will be useful to you thanks.