ZOOM PASCAL

YOU CAN COUNT ON
Abacus Software

==

ZOOM PASCAL

(C)1983 BY WILLI KUSCHE

==

Complete Pascal Software Development

Package for the Commodore 64

==
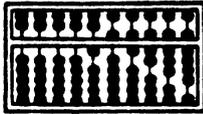
Abacus Software

P.O. Box 7211

Grand Rapids, MI 49510

PREFACE

With the advent of the affordable home computer, Pascal has
become a very popular language. In the past four years we
have seen a tremendous offering in both Pascal compilers and
application software written in the Pascal langauge.

The Commodore 64 has become one of the most sought after
micros because of its tremendous features at an unbelievably
low price. And the number of high quality software offerings
for the Commodore 64 has skyrocketed in the past six months.
We believe that ZOOM Pascal 64 is one such package. It has
excellent features at a very reasonable price.

Although ZOOM Pascal 64 is a subset of standard Pascal, it
contains extensions and features that make it a very useful
software package. In addition to the full string handling
functions, there are several machine language oriented
features that make it very useful: hex number input and
output; byte anding and oring, file handling procedures and
function.

The most attractive feature of ZOOM Pascal 64 is that it
creates true 6502 machine code. This make ZOOM Pascal 64
extremely fast when compared to BASIC or other Pascal
compilers. Only one other Pascal compiler creates true 6502
machine code, KMMM Pascal by WILSERV Industries.

In fact, ZOOM Pascal 64 and KMMM Pascal were written by the
same person, Willi Kusche. ZOOM Pascal 64 is thus a subset
of KMMM Pascal. If you are interested, KMMM Pascal also
includes these features: RECORDS for creating structured
data types; pointers and procedures for dynamic allocation;
fast EDITOR/COMPILER to speed up program development.

KMMM Pascal may be ordered directly from:

WILSERV INDUSTRIES
P.O. Box 456
Bellmawr, NJ 08031
609/227-8696

In the meantime, ZOOM Pascal 64 should provide you with a
complete Pascal package. It is suitable for learning this
fascinating language as well as for program development. You
should note that programs compiled under ZOOM Pascal 64 may
be distributed without royalties.

We thank Willi Kusche for his cooperation in making ZOOM
Pascal 64 available.

Arnie Lee
Grand Rapids, MI
September 29, 1983

TABLE OF CONTENTS

1.0   INTRODUCTION

ZOOM Pascal 64 is a true compiler. It generates 6502 machine language from a Pascal source file. Most versions of Pascal use some form of interpreter. This explains why, in timing tests, ZOOM Pascal 64 was the fastest version.

ZOOM Pascal 64 is a subset of the Pascal described by Jensen and Wirth in their book, which is the 'bible' for Pascal. It is also a subset of and supplies all of the string functions of UCSD Pascal. The section **Differences from standard Pascal** explains the major differences between ZOOM Pascal 64 and standard Pascal.

This manual is only a guide to use of ZOOM Pascal 64 and is not intended to teach the user how to write programs in Pascal. Consult one of the references in the Appendix C - the bibliography for more information about Pascal programming.

ZOOM Pascal 64 runs on standard Commodore 64 with a 1541 disk drive. Although ZOOM Pascal 64 can work with a cassette drive, use of a cassette drive is suitable only for small source or data files. ZOOM PASCAL 64 is therefore not distributed on cassette and ABACUS Software does not supɪ ort a cassette version.

For those of you who are unfamiliar with compilers, you must follow several steps before you can RUN a Pascal program:

   1) First you must create the source program written in the high-level Pascal language. You use the **EDITOR** (ZE-64 on the distribution diskette) to create and or modify your source program.
   2) Next you compile this source program into an intermediate form in memory called P-code. The **COMPILER** (ZC-64 on the distribution diskette) checks source program for grammar errors and semantics and notifies the user of any such errors.
   3) If the source program compiles without error, then the **TRANSLATOR** (called ZT-64 on the distribution diskette) converts the P-code into true 6502 machine language. When the **TRANSLATOR** is finished, the program is now ready to RUN or SAVE to disk. This **object** program is a machine language equivalent to the high level Pascal source program which you wrote in step 1.

Compiled programs are compatible with Commodore BASIC. They are loaded into memory and executed as if they were BASIC programs, even though they are written in machine language. Compiled Pascal programs may be distributed without any royalties to the author.

1.1 DISTRIBUTION DISKETTE

The distribution diskette contains the following:

* ZE-64   - EDITOR for creating Pascal source files
* ZC-64   - COMPILER for compiling the source file to intermediate P-code
* ZT-64   - TRANSLATOR for translating the inter- mediate P-code to executable 6502 machine language
* PCCEXMP - sample source program to read and write to the disk command channel.
* PLSTCBMD - sample source program to access the diskette directory
* PRANDCR - sample source program to create a random file on diskette
* PRANDUPD - sample source program to update a random file on diskette
* PSIEVE   - sample source program to find all prime numbers between 1 and 1000.
* BSIEVE   - sample BASIC program equivalent to PSEIVE to demonstrate the difference in execution between compiled Pascal and interpreted BASIC.

The EDITOR and TRANSLATOR are unprotected. You should copy them to a separate disk immediately. To do this:

1) FORMAT A BLANK DISKETTE by inserting a new diskette into the disk drive and typing:

   OPEN1,8,15:PRINT#1,"N:name,xx":CLOSE1

   where **name** is any 16 character diskette name and **xx** is a two-digit diskette identifier

2) COPY THE EDITOR by inserting the distribution diskette into the drive and typing:

   LOAD "ZE-64",8<RETURN>

   After the EDITOR is loaded, replace the distribution diskette with the newly formatted diskette and type:

   SAVE "ZE-64",8<RETURN>

3) COPY THE TRANSLATOR by inserting the distribution diskette into the drive and typing:

   LOAD "ZT-64",8<RETURN>

   After the TRANSLATOR is loaded, replace the

distribution diskette with the newly formatted
diskette and type:

SAVE "ZT-64",8<RETURN>

4) The Pascal Compiler **ZC-64** is protected in order
   to safeguard the copyright of this software
   package. It cannot be copied by normal computer
   equipment. Therefore you will have to use the
   distribution diskette when compiling your source
   programs.

The next part of the manual, SECTION 1.2, shows you how to
compile a sample Pascal source program contained on the
distribution diskette.

1.2 COMPILING A SAMPLE PROGRAM

To become familiar with ZOOM Pascal 64, we suggest that you
compile and translate several of the sample source programs
contained on the distribution diskette. Throughout this
section only, you should use the distribution diskette,
since the sample source files are on it. When the compiler
asks you to REMOVE THE DISTRIBITION DISKETTE, leave it in
the drive and press the <RETURN> key.

NOTE after you compile the sample program: When later
writing your own Pascal source programs, use a diskette
containing a copy of the EDITOR and TRANSLATOR as described
in SECTION 1.1. Use the distribution diskette only to load
the COMPILER.

1) Type: LOAD "ZC64",8 <RETURN>

2) After the compiler is loaded, type: RUN <RETURN>

3) The copyright message displays and a short time
   after the computer asks you to:

            REMOVE DISTRIBUTION DISKETTE
            PRESS RETURN WHEN READY

   Do not remove the distribution diskette. Instead
   just press the <RETURN> key.

4) The computer asks: SOURCE FILE NAME?

   Type: PSIEVE <RETURN>

5) The computer asks: TEXT LITERALS IN SOURCE?

   Type: N <RETURN>

6) The computer asks: USE PRINTER?

   Type: N <RETURN>

   The source program now compiles to Pcode in memory.

7) When it is complete the computer asks:

            PRESS RETURN WHEN READY

    Simply press the <RETURN> key.

8) The computer asks: EXECUTE TRANSLATOR?

   Type: Y <RETURN>

   The Pcode is now converted to 6502 machine code.

4

9) When completed type: **RUN <RETURN>**

   The SIEVE program runs and displays all of the prime number between 1 and 1000.

10) You can save the compiled SIEVE program by typing:

   SAVE "SIEVE",8 <RETURN>

11) To compare the speed of the compiled program to a BASIC version of the SIEVE, you can run the BASIC program BSIEVE.

The remainder of this manual is divided into three major parts.

Part 2 describes the EDITOR which is used to create and modify your Pascal source program.

Part 3 describes the COMPILER/TRANSLATOR syntax and rules for writing programs under **ZOOM Pascal 64**.

Part 4 contains the appendices.

2.0 THE EDITOR

The EDITOR is used to create and maintain your Pascal language source program.

The EDITOR has two modes of operation.

First, there is the COMMAND MODE. The COMMAND MODE is used primarily to control file access, but can also be used to edit a Pascal program. In the COMMAND MODE, the EDITOR operates by reacting to commands which you enter. You know you are in the COMMAND MODE when you see a blinking cursor next to an asterisk. A command consists of a single or double letter abbreviation which specifies the operation to be performed.

Second, there is the WINDOW MODE. The WINDOW MODE is used as a full screen editor. You enter the WINDOW MODE when you type a W command from COMMAND MODE. The EDITOR, in effect, opens a window into the text buffer. The EDITOR transfers to the screen as many lines of the text buffer as the screen can hold, beginning with the line pointed to by the current position of the character pointer. You can use the cursor positioning keys to move the solid cursor around the screen. If you leave the screen boundary by pressing the cursor up key while at the top of the screen or pressing the cursor down key while at the bottom of the screen, you scroll the window through the text buffer. The STOP key returns the EDITOR to the COMMAND MODE.

Users whose only exposure to an editor is the one that is part of BASIC, should make sure that they understand the next paragraph. This is because the <RETURN> key works quite differently while the EDITOR is running.

Before starting the EDITOR, the user must decide on a character to be used as a command separator or escape character. We recommend you use the exclamation point. This manual assumes that you have chosen the character ! as the escape character. After entering the escape character the EDITOR prompts you with an asterisk, indicating that it is ready to a series of commands (command string) indicates that the EDITOR has executed the previous command string and is ready to accept another command string.

2.1 OPERATING INSTRUCTIONS

The EDITOR is named **ZE-64** on the distribution diskette.

You may copy the EDITOR to a backup diskette by following the directions in the Section 1.1 (DISTRIBUTION DISKETTE).

To run the EDITOR do the following:

1) Type:

    LOAD "ZE-64",8 <RETURN>

2) After the EDITOR is loaded type:

    RUN <RETURN>

3) After printing a copyright statement, the EDITOR prompts the user for the **escape** character. Type:

    !   (no <RETURN> key necessary)

4) The EDITOR now displays an asterisk indicating that it is ready to accept a command. The user may then enter a single command or a series of commands (command string). Two consecutive escape characters (!!) terminate the command string. Appearance of an asterisk after entry of a command indicates that the EDITOR has executed all of the previous commands and is ready to accept another command string.

You may use the DELete key is the only way to correct keying errors when entering commands. However the EDITOR is not very bright when you delete a carriage return in the command string.

The EDITOR echos the space key as an underline character when accepting command string characters. This aids the user in determining the position of the cursor.

The H command terminates the EDITOR and returns control to BASIC.

2.2 CREATING A Pascal SOURCE FILE

This section assumes that you have just started the EDITOR
and are using the exclaimation point (!) as the escape
character.

Press the I key once. The letter I appears next to the
prompting asterisk and the blinking cursor is now to the
immediate right of the letter I.

Now press the A key followed by the <RETURN> key. Repeat
this two key sequence for each letter of the alphabet from B
to X. Finally, press the escape character key twice (!!).

At this point, the characters *IA should be on the top line
of the screen, the characters !!* on the bottom line of the
screen and the letters B through X on the lines between,
with each letter on a separate line. If this is not the
case, press the H key once and the escape character key
twice to return to BASIC. Then enter the BASIC command RUN
to restart the EDITOR and go back to the beginning paragraph
of this section.

You may have noticed that nothing happened as you pressed
each key (except the last), other than having the key echoed
on the screen. This is because the EDITOR merely stored each
key stroke in a command buffer. Since the first letter
stored was the letter I, you were entering the INSERT
command. Everything following the letter I, up to but not
including the escape character was stored in the text
buffer. The INSERT command was not actually executed until
the second consecutive escape character was found.

Now, let's take a look at the text buffer via the window
mode. To do this, press the following five keys: B!W!!. The
B gets you to the beginning of the text buffer. The W gets
you into the WINDOW MODE of the EDITOR.

Take a moment to note some of the characteristics of the
WINDOW MODE. The bottom line in white is the status line. It
shows whether the screen is in the text or graphics mode. A
carriage return in the text buffer displays as a back arrow
character on the screen (<-). Finally, the cursor is a solid
white block, instead of a blinking white block.

Now press the cursor right, cursor down, cursor left and
cursor up keys. The solid cursor should be in the upper left
hand corner of the screen. If it isn't, press the HOME key.

Now make space for ten new lines by pressing the INSERT LINE
key ten times. The INSERT LINE key is the F1-key. Next,
press the zero key bollowed by the <RETURN> key. Repeat this
sequence for the digits zero to line and the letters A to N,
each on a separate line.

Now press the **CLR** key (shifted **HOME** key). This moves the solid cursor to the bottom line of the window. Next press the cursor down key a few times. Each time you do, a line scrolls off the top of the screen and a new line appears at the bottom. If you press the cursor down key often enough and the last line of the text buffer is displayed, then further depressions of the cursor down key causes the cursor to fill with white lines.

To get back to the beginning of the text buffer, press the **HOME** key. Then press the cursor up key until the screei no longer changes.

Finally, delete the lines with digits on them by press the **DELETE LINE** key ten times. The **DELETE LINE** key is the F3-key.

To store the file onto disk, you must return to the **COMMAND MODE**. To return to the **COMMAND MODE**, simply press the **STOP** key.

The command to write the text buffer to disk is:

    GWO:LETTERS!P!GC!!

The **GW** letter paid is the command to open an ouptut file and is mnemonic for "Get ready to Write".

The **GW** letter pair must immediately be followed by the output file name, which in turn, must have a device specifier. Thus the file name is **LETTERS** and the drive is drive 0.

The **P** command actually writes the text buffer to the disk and is mnemonic for "Put".

The **GC** command closes the output file.

Notice that the escape character separates commands and the double escape character (!!) tells the EDITOR to perform the entire command string immediately.

2.3   UPDATING A Pascal SOURCE FILE

This section assumes that you have just started the EDITOR
and are using the exclamation point as the escape character.
It also assumes that you have successfully created a file
named LETTERS on the diskette after following the
instructions in the prior section.

To create a revised version of a file on diskette, use the
UY command. This is a mnemonic for "Update and Yank" file
into memory. Therefore, to update the LETTERS file created
previously, use the command string:

   UYO:LETTERS!W!!

This command string brings the LETTERS file into the text
buffer and puts the EDITOR into the **WINDOW MODE**. After
making any desired changes in the window mode, drop back to
the **COMMAND MODE** by pressing the STOP key. To complete the
file update, use the letter pair **UE** which is mnemonic for
"Update End".

To copy a file from one diskette to another, an alternate
series of commands must be used. Therefore, to read the
LETTERS file created previously, use the command:

   GRLETTERS!Y!GC!!

The **GR** letter paid command is mnemonic for "Get ready for
Read" and must be immediately followed by the name of the
file to read.

The **Y** command reads the file opened by the GR command
and is mnemonic for "Yank" file into memory.

The **GC** letter pair command closes the input file so that the
diskette may be removed from the drive.

After placing the diskette to receive the output file into
the drive, use the command string:

   GWO:NEWNAME!P!GC!!

The **GW** letter paid is the command to open an output file and
is mnemonic for "Get ready to Write".

The **GW** command must be immediately followed by the output
file name which, in turn, must have a device specifier.

The **P** command actually writes the file stored in the text
buffer to the diskette and is mnemonic for "Put".

The **GC** command closes the output file.

10

2.4   THE WINDOW MODE

In the **WINDOW MODE**, the commands valid in **COMMAND MODE** have
no effect. The cursor left, cursor right, cursor up, cursor
down, character insert, character delete, home and shifted
return keys react the same way as they do in BASIC.

The <RETURN> key replaces the character under the solid
cursor with a substitute carriage return character (<-),
clears the rest of the line to spaces and positions the
solid cursor at the beginning of the next line. Any other
key, which is not a control key, replaces the character
under the solid cursor and moves the solid cursor one
position to the right.

The **CLR** key does not clear the screen, but instead moves the
solid cursor to the bottom of the screen.

Two keys are used to delete or insert lines. The F1-key is
used to insert a blank line at the cursor position. The F3-
key is used to delete a line at the cursor position.

Normally, pressing a key such as the letter A replaces a
character on the screen. It is possible to active an insert
mode within the window mode so that any characters are
automatically inserted in a line, rather than replacing
characters in a line. To activate the insert mode, press the
F7-key followed by the **ISRT/DEL** key. The words **INSERT MODE**
appears on the status line. The **STOP** key is used to
terminate the insert mode.

Any control keys which are not specifically mentioned in
prior paragraphs are ignored in the window mode.

The **STOP** key returns you to the command mode. Since the **STOP**
key is also used to terminate the insert mode withing the
window mode, you must press the **STOP** key twice to return to
the command mode if the insert mode is active.

2.5   THE COMMAND MODE

To understand the **COMMAND MODE** of the EDITOR, it is necessary to introduce the concept of the **character pointer**. Since Pascal source programs do not have line numbers, we need to have another way of telling the EDITOR which line or lines we wish to work on. For this purpose, there is a **character pointer** to which the user can refer.

There are many different commands in command mode. They are listed alphabetically:

B   - move character pointer to Beginning of text
                  buffer

Csss!ttt!   - change character(s) in text buffer
                  The change command must be followed by two
                  text strings. Each text string must be
                  terminated by the escape character. The
                  first string sss may not be longer than 80
                  characters. The second string ttt may be any
                  length. The EDITOR searches the text buffer
                  for an exact match of string **sss**. If it
                  finds a match, it replaces it with the new
                  string **ttt**. If the second string is a null
                  string, then the change command acts as a
                  search and delete command. After completion,
                  the character pointer is immediately to the
                  right of the last character changed.

nD   - delete character(s) in text buffer
                  The character immediately to the right of
                  the character pointer is removed from the
                  text buffer. If the D command is preceeded
                  by a repeat value n, then n characters to
                  the right of the character pointer are
                  removed from the text buffer.

FD   - Flip Display
                  This command toggles the screen between
                  graphics mode and text mode.

FM   - Flip Memory
                  This command examines every character in the
                  text buffer. Any character in the range of A
                  to Z is changed to its lower case
                  equivalent. Any character in the range of a
                  to z is chaanged to its upper case
                  equivalent.

GC   - close file(s)

GRd:fff   - open input file (Get ready for Read)

GWd:fff   - open ouput file (Get ready for Write)

H   - Halt (exit editor)

I   - Insert character(s) into text buffer
                  All text must be entered using the I
                  command. This command must be followed by
                  at least one character to be inserted. All
                  characters following the I up to but not
                  including the escape character, are stored
                  in the text buffer. The text string to be
                  inserted may be of any length and may
                  include any number of carriage return
                  characters. The point of insertion is the
                  current position of the character pointer.
                  After completion of the insert command, the
                  character pointer is immediately to the
                  right of the last character inserted.

nJ   - move character pointer to specific line in
                  text buffer
                  This command moves the character pointer to
                  a specific line in the text buffer. The J
                  command (Jump) must be preceeded by a number
                  which is the line number to move the
                  character pointer to.

nK   - Kill delete remainder of line or entire
                  line(s) in text buffer
                  This command deletes all characters to the
                  right of the character pointer, up to and
                  including the first occurance of a carriage
                  return character ($0D) from the text buffer.
                  If the K command is preceeded by a repeat
                  value n, then n lines to the right of the
                  character pointer are removed from the text
                  buffer.

nL   - move character pointer within text buffer by
                  Line(s)
                  This command moves the character pointer by
                  lines. The character pointer is moved to the
                  left until a carriage return character ($0D)
                  is found. The character pointer is set to
                  point immediately to the right of the
                  carriage return (beginning of the current
                  line). If the L command is preceeded by a
                  repeat value n, then a search for the nth
                  carriage return character to the right of
                  the current position of the character
                  pointer is initiated. If the repeat value is
                  preceeded by a minus sign, then the search
                  is to the left of the current position of
                  the character pointer, that is, towards the
                  beginning of the text buffer.

13

nM   -   Move character pointer within text buffer by
character(s)
This command moves the character pointer by
characters. The character pointer is moved
one character to the right, that is towerds
the end of the text buffer. If the M command
is preceeded by a repeat value n, then the
character pointer is moved n characters to
the right. If the command is preceeded by a
minus sign, then the character pointer is
moved to the left, that is, towards the
beginning of the text buffer.

P   -   write output file from text buffer (PUT)
This command writes a diskette file from the
text buffer. This command must be preceeded
by a GW command.

Ssss!   -   Search for character string in text buffer
This command searches for a string sss. All
character following the S, up to but not
including the escape character are used in
an exact match of the text buffer. The
search text may not be longer than 80
characters. The search begins with the
character immediately following the currenet
position of the character pointer and
continues to the end of the text buffer. If
an exact match is found, the command ends
with the character pointer immediately to
the right of the last matching character in
the text buffer. If no match is found **STBING
NOT FOUND** is printed and the character
pointer is left at the beinning of the text
buffer.

nT   -   Type; display lines within text buffer.
This command displays text from the text
buffer. If only T is entered, the character
pointer is moved to the beginning of the
text buffer and the entire text buffer is
displayed on the screen. After completion of
the display, the character pointer remains
at the beginning of the text buffer. If the
command is preceeded by a repeat value n,
then only n lines of text are displayed
beginning at the current position of the
character pointer. If the character pointer
is not at the beginning of a line, only that
postion of the line to the right of the
character pointer is displayed. When only
part of the text buffer is displayed, the
position of the character pointer is not
changed.

UA   -   Update Abort; cancel outstanding file update
                  This command cancels any outstanding file
                  update without rewriting the file.

UDfff   -   Utility Delete; delete file from diskette
                  This command delete a file on diskette. The
                  file fff is the name of the file to be
                  deleted.

UE   -   Update End; write text buffer and close file
                  This command ends the update process by
                  rewriting the updated file to the diskette.

UI   -   Utility Initialize; initialize drive

URfff!ggg   -   Utility Rename; rename a file on diskette
                  This command renames a file on a diskette.
                  The first file fff is the existing file and
                  the second file ttt is the new file name.

UYd:fff   -   Update Yank; open and fill text buffer from
                  file.
                  This command opens the file fff on device d
                  to be updated and reads it into the text
                  buffer.

W   -   enter Window mode

                  COMMANDS FROM WINDOW MODE

                  INSERT LINE   F1KEY
                  DELETE LINE   F3KEY
                  ENTER INSERT MODE   F7KEY & ISRT/DEL key
                  LEAVE INSERT MODE   STOPKEY
                  EXIT WINDOW MODE   STOPKEY
                  CURSOR MOVEMENT KEYS   same as BASIC
                  HOME CURSOR   HOMEKEY
                  BOTTOM CURSOR   CLRKEY

X   -   eXecute macro command

XM   -   define Macro command

X?   -   display macro command

Y   -   Yank; read input file into text buffer

Z   -   move character pointer to the end of the
                  text buffer

.   -   display the current character pointer within
                  the text buffer

:   -   display the number of lines stored in the
                  text buffer

2.6 ASCII FILES

The EDITOR gives you the option of deciding whether or not
to translate to ASCII when storing the text buffer to an
external file or translating from ASCII when filling the
text buffer from an external file.

The translate mechanism is controlled by the state of the
screen mode. If you are not sure which mode the screen is
in, switch to the WINDOW MODE where the screenmode is always
displayed on the bottom status line.

If your machine is in graphics mode, no translation takes
place when reading into or writing out of the text buffer.

If you machine is in text mode, translation takes place when
executing a Y or a UY command. The EDITOR translates the
characters of the external file from ASCII to PET ASCII
format. Similarly translation occurs, if your computer is in
text mode and you execute a P or UE command. The EDITOR
translates the characters in the text buffer from PET ASCII
to ASCII before writing them to an external file.
od
The FD command is the only way to change the screen mode of
your computer when the EDITOR is active. The FD command is a
flip-flop command in that it changes the screen mode from
text to graphics or vice versa every time it is executed.
When an editing session is terminated, the EDITOR restores
the screen mode to the mode that was in effect when the
EDITOR was started.

3.0   THE COMPILER/TRANSLATOR

After you have used the editor to create or modify your
Pascal source program you use the COMPILER/TRANSLATOR to
convert this high level language program to machine code.

The following instructions will detail the operation of the
COMPILER/TRANSLATOR.

3.1 OPERATING INSTRUCTIONS

To begin, run the COMPILER by doing the following:

1) Type;

> **LOAD"ZC-64",8 <RETURN>**

2) After the COMPILER is loaded type:

> **RUN <RETURN>**

3) The remainder of the COMPILER is now loaded. The screens displays a copyright message as the software is loaded. The disk drive makes some strange noise as loading nears completion, but this is normal. Then the compiler prompts you to:

> **REMOVE DISTRIBUTION DISKETTE**
> **PRESS ANY KEY WHEN READY**

4) You should have previously created or modified you source program on a diskette other than the distribution diskette. Therefore you now remove your distribution diskette from the drive and insert the diskette containing your Pascal source file. After changing diskettes press the <RETURN> key.

5) The screen displays a short identification and then ask to to enter the:

> **SOURCE FILE NAME?**

Type in name of the file to be compiled and then press the <RETURN> key.

If the source file is not contained on the diskette, then the COMPILER prompts you for another source file name.

6) The screen displays another message:

> **TEXT LITERALS IN SOURCE?**

If you enter Y, the screen is shifted to text mode; entry of any other character leaves the screen in graphics mode.

7) The screen then displays the message:

USE PRINTER?

If you enter Y, the source listing is directed to a printer; entry of any other character directs the source listing to the screen.

If the source listing is directed to a printer, there are two more prompts:

a)   DEVICE NUMBER?

If you have a normal printer type 4 and the <RETURN> key. Otherwise enter your printer device number.

b)   GENERATE LINE FEED?

Enter a Y if your printer does not automatically generate a line feed with a carriage return.

If the source listing is directed to a printer, each source line is preceeded by two numeric values. The first number is a line number, which can be later used to edit the source file. The second number is the number of the first P-code generated by the corresponding source statement.

8) The source program is now compiled.

If an error is found, an up arrow is displayed under the source statement that caused the error. An error message describing the error is also displayed. You must use the EDITOR to correct an error in the source file in such a case. Reset your computer before rerunning the EDITOR.

9) If no errors are found the compiler asks you to:

PRESS RETURN WHEN READY:

This gives you an opportunity to read the remainder of the source listing on the screen.

10) The compiler then prompts you with:

EXECUTE TRANSLATOR?

If you enter Y, then the TRANSLATOR is automatically loaded and run. Entering any other character allows you to compile another program.

NOTE: Be sure that a copy of the TRANSLATOR (called

ZT-64) was saved to your source file diskette as
described in SECTION 1.1.

The TRANSLATOR makes two passes through the Pcode.
In a first pass, the TRANSLATOR counts the number
of Pcodes referred to by other Pcodes and displays
this count as n LABELS.

In a second pass, the TRANSLATOR generates the 6502
machine code and displays the statistics on the
screen.

The TRANSLATOR generates a machine language program
consisting of an 8K package of support subroutines
and the machine code generated from the Pascal
source statements. The combined program is
preceeded by a SYS statement which allows the
program to be executed like a normal BASIC program.

11) After the TRANSLATOR is finished, you can test the
compiled program by typing:

RUN <RETURN>

12) You can also save your compiled program to disk by
typing:

SAVE"progname",8 <RETURN>

You may now run this program on any other Commodore
64 without having to compile it again. It is truly
a machine language version of your original Pascal
source program.

3.2 DIFFERENCES FROM STANDARD PASCAL

ZOOM Pascal 64 is not a full Pascal compiler. This section describes those features of standard Pascal that are not available with ZOOM **Pascal 64**. It also describes those features of ZOOM **Pascal 64** which are extension to standard Pascal. This section is designed to give you a quick look at ZOOM **Pascal 64** features. Subsequent sections give you more detail about the langauge elements.

A Pascal program consists of a program heading and a block. A block consists of five declaration parts and a statement part.

The program heading is optional for ZOOM **Pascal 64** programs.

The five declaration parts of a block are:

  * LABEL declaration part
  * CONSTant declaration part
  * TYPE declaration part
  * VARiable declaration part
  * PROCEDURE and FUNCTION declaration part

The **LABEL** declaration part is not recgonized by ZOOM **Pascal 64**. Therefore you may not use any **LABEL** declarations with ZOOM **Pascal 64** programs.

The CONSTant declaration part is compatible with standard Pascal.

The **TYPE** declaration part recognizes only scalars and subranges. You may not use the reserved words **FILE**, **RECORD**, **SET** or pointer types. You may use the standard identifier **TEXT** which is a standard type predeclared as a **FILE OF CHAR**. Refer to the section **INPUT/OUTPUT PROCEDURES** for more details.

The **VARiable** declaration part recognizes all keywords. However the reserved word **ARRAY** may contain only one dimension. You may also use the type **STRING** which is an extension to standard Pascal.

The **PROCEDURE** and **FUNCTION** declaration part does not recognize the reserved words **PROCEDURE**, **FUNCTION** or VAR in the procedure or function header.

In the statement part:

  * the reserved words WITH, GOTO, and **IN** are not recognized.
  * the functions EOF, EOLN and ROUND are not recognized
  * the reserved words **MEM**, **CALL**, **SHL** and **SHR** are

21

extensions

The reserved word **MEM** may appear as a factor is an expression or may appear to the left of the assignement symbol (:=), in order to allow access to any memory location.

The reserved word **CALL** is procedure to allow you to execute a machine language routine.

The reserved words **SHL** and **SHR** may appear as operators in a term, to allow bit shifting.

You may use a special character **$** in **ZOOM Pascal 64**. The **$** may be used in two different ways:

1) When preceeding hexadecimal digits, it defines a hexadecimal constant
2) When a suffix to a variable or expression in a **READ, READLN, WRITE** or **WRITELN** procedure, four hexadecimal digits are exptected as input or written as output.

ZOOM Pascal 64 provides a number of non-standard functions and procedures. The functions are: **ANDB, NOTB, ORB, RND, GETKEY, INKEY, CONCAT, COPY, DELETE, LEFTSTR, LENGTH, POS** and **RIGHTSTR**. The procedures are: **CLOSE** and **EXIT**.

3.3 PROGRAM STRUCTURE

Every Pascal programs consists of a heading and a a block.

A block, in turn, consists of a declaration part and a statement part.

The decla part is further subdivided into five parts; the label declartion part; the constant declaration part; the type declaration part; the variable declaration part; and the procedure and function declaration part.

The following sections are organized the order that they would appear in a program:

    3.3.1   PROGRAM HEADING
            DECLARATION part
    3.3.2   LABEL declaration part
    3.3.3   CONSTant declaration part
    3.3.4   TYPE declaration part
    3.3.5   VARIABLE declaration part
    3.3.6   PROCEDURE and FUNCTION declaration part
    3.3.7   STATEMENT part

3.3.1 PROGRAM HEADING

A Pascal program consists of a program heading and a program block. To allow compatibility with other version of ZOOM Pascal 64, a program heading is NOT required.

The COMPILER scans for the reserved word **PROGRAM** at the start of the Pascal source file. If it is not present, the COMPILER displays the message **HEADER MISSING**, but continues with the compilation.

If the Pascal source file does contain the reserved word PROGRAM, the COMPILER expects an identifier to follow. The COMPILER bypasses the words and characters following up to and including a semi-colon.

Examples of valid program headings are:

> PROGRAM SAMP1;
>
> PROGRAM SAMP2(INPUT,OUTPUT);
>
> PROGRAM PTEST(description,date);

3.3.2 LABEL DECLARATIONS

ZOOM Pascal 64 does not recognized the label declaration part. Therefore the **GOTO** reserved word cannot be used.

The non-standard procedure **EXIT**, if used properly can be used in place of a label.

3.3.3 CONSTANT DECLARATIONS

There are no restrictions in **ZOOM Pascal 64** regarding the constant declaration part. However, the COMPILER accepts the dollar sign character (**$**) as a prefix to hexadecimal constants.

If a **$** is followed by two hexadecimal digits, then the constant is declared to be type **CHAR**. If a **$** is followed by four hexadecimal digits, then the constant is declared to be type **INTEGER**. If a **$** is followed by any other number of characters, an error results.

Please note that this extension is often handy, but is not supported by standard Pascal. It should not be used in programs that are to be moved to other systems.

The following are examples of constant declarations:

```
CONST ONE = 1;              (* type is INTEGER        *)
      HALF = 0.5;           (* type is REAL           *)
      THREEQUARTERS = .75;  (* leading zero missing   *)
      HALFMILLION = 0.5E+6; (* type is REAL           *)
      LETTERA = 'A';        (* type is CHAR           *)
      STARTSTRING = 'START';(* type is STRING         *)
      CLRSCREEN = $93;      (* type is CHAR           *)
      C64STATUS = $0090;    (* type in INTEGER        *)
```

3.3.4 TYPE DECLARATIONS

ZOOM Pascal 64 supports scalars and subranges. As a matter of fact, if you declare a variable to be a subrange or a scalar, you MUST use the **TYPE** declaration to define the subrange or scalar. You may NOT declare any structured types.

Examples of valid TYPE declarations are:

```
TYPE BYTE=0..255;
     COLOR=(RED, YELLOW, BLUE);
     RANGE=1..50;
VAR  CHARACTER: BYTE;
     CARPAINT: COLOR;
     VECTOR: ARRAY[RANGE] OF REAL;
```

Example of invalid TYPE declarations are:

```
 TYPE ARRAYTYPE=ARRAY[1..50] OF REAL; (* structured
                                  not allowed)
 VAR CHARACTER: 0..255;        (* must be declared
                                  in TYPE        *)
     COLORVAR:(RED, YELLOW, BLUE); (*  ditto     *)
```

3.3.5 VARIABLE DECLARATIONS

ZOOM Pascal 64 allows variable to be simple types or an array of simple types. The standard simple types are:

>   BOOLEAN
>   CHAR
>   INTEGER
>   REAL
>   TEXT

TEXT is a standard Pascal shorthand for the declaration **FILE OF CHAR**. In addition, a user defined type may be specified. Remember that the type definition part may only specify scalars or subranges.

As mentioned in **TYPE DECLARATIONS**, arrays may have only a single dimension. The reserved word **PACKED** may preceed the reserved **ARRAY**, but is treated as a comment.

ZOOM Pascal 64 also allows a variable to be declared to be of type **STRING**. In the variable declaration, the word **STRING** may, optionally, be followed by a numeric literal enclosed in brackets, which specifies the maximum length of the string variable. If length is not specified, the length defaults to 80 characters. The maximum length of a string is 196 characters.

Examples of valid STRING VARIABLE declarations are:

```
VAR FIRSTNAME: STRING[12];
    LASTNAME:  STRING[25];
    ADDRESS1, ADDRESS2: STRING[30];
    CITY:      STRING[20];
    STATE:     STRING[20];
    ZIP:       STRING[5];
    LAZY:      STRING;           (* SAME AS STRING[80] *)
```

You may also have arrays of string variables such as:

>   **ARRAY[0..n] OF STRING[length]**

but the maximum length of the string is 127 in an array. Also, the maximum length of a string that is passed into a functionor procedure, or returned from a function is 80.

In order to optimize the run time speed of the generated machine language program, NO check is made to insure that the length of a string does not exceed the declared length. This can definitely cause problems when a compiled program is run. Therefore, do not be stingy when allocating the length of variable of type **STRING**.

3.3.6 PROCEDURE AND FUNCTION DECLARATIONS

A procedure is a subroutine that is activated by a procedure statement. A function is a subroutine that returns a value and therefore can be used as a factor in an expression.

The declaration of either a procedure or a function must begin with a heading. The heading serves to identify the number and types of parameters to be passed to the procedure or function. Additionally, the function heading identifies the type of the value returned by the function.

Standard Pascal allows the parameters passed to a procedure or function to be either value, variable, procedure or function paramaters. ZOOM Pascal 64 allows only value parameters to be passed to procedures or functions.

Examples of PROCEDURE and FUNCTION declarations are:

    PROCEDURE UPLOW(data:STRING);

    FUNCTION SQUARED(numb:REAL):REAL;

3.3.7 STATEMENT PART

The statement part of a Pascal program may be either a simple statement or a compound statement. A compound statement is a series of simple statements separated by a semi-colon character and bounded by the reserved words **BEGIN** and **END**.

A statement may be either an assignement statement, a procedure statement, a repetitive statement (**WHILE, REPEAT** of **FOR**), a conditional statement (**IF** or **CASE**), a GOTO statement or a **WITH** statement. ZOOM **Pascal 64** does not support the **GOTO** and **WITH** statements.

An assignment statement has the following format:

        variable := expression

The variable and the expression must be of the same type.

Standard Pascal allows two exceptions to this rule. The first is that if the type of the variable is **REAL**, then the type of the expression may be **INTEGER** or a subrange thereof. The second is that the type of the expression may be a subrange of the variable or vice-versa.

ZOOM **Pascal 64** allows these exception, plus two others. If the variable is of type **STRING**, then the expression may be of type **CHAR**. In this case, an automatic type conversion occurs. If the variable is of type **CHAR**, then the expression may be of type **STRING**. In this case, the first character of the string is extracted and an automatic type conversion occurs.

ZOOM **Pascal 64** also allows a non-standard reserve word, **MEM**, to be the variable in an assignement statement. The reserved word **MEM**, must be subscripted by an integer expression. The expression to the right of the := must be of type **CHAR**. This statement has the same function as a **POKE** in BASIC. An example of this is:

        MEM[$0400] := 1; (* PLACE AND 'A' ON C64 SCREEN *)

3.4 RESERVED WORDS

The standard Pascal reserved words are listed in Appendix A.
Of those reserved words, ZOOM Pascal 64 does not recognize
the following words: FILE, GOTO, IN, LABEL, NIL, RECORD, SET
and WITH.

Additionally, ZOOM Pascal 64 has defined four words as
reserved words. These are CALL, MEM, SHL and SHR.

Use of the MEM reserved word in a statement equivalent to a
BASIC POKE is explained in the previous section. The MEM
reserved word may also appear as a factor in an expression,
to serve as an equivalent to a BASIC PEEK. The word MEM must
be followed by an integer expression enclosed by brackets.
Examples are:

     STATUS := ORD(MEM[$0090]);   (* STATUS FOR C64 *)

The reserved words SHL and SHR may appear as a multiplying
operator between two factors in a term of an expression. An
integer factor to the left of SHL is shifted left by the
number of bits represented by the value of the integer
factor to the right of the SHL. Similarly, SHR causes a
shift right.

The following example shows how to store an address value in
memory:

     MEM[STOREADDR] := CHR(ADDRVALUE); (* STORE IN LOW HALF *)
     MEM[STOREADDR+1] := CHR(ADDRVALUE SHR 8); (* STORE HIGH
                                             HALF   *)

3.5 P ECLARED IDENTIFIERS

In addition to the reserved words defined by standard Pascal, there are words which are predeclared by standard Pascal. They differ from reserved words in that you may redefine them for your own use. If you do so, then your definition of these standard identifiers replaces the defined usage in Pascal. However, doing so is not recommended.

The complete list of standard identifiers appears in Appendix B. All of the standard function identifiers have been implemented except for EOF, EOLN and ROUND. The EOF function may be emulated by using the MEM function to examine the location mentioned in the prior section. The EOLN function may be emulated by testing for a carriage return character (* $0D *). The ROUND function may be emulated by including the following function in your source file:

    FUNCTION ROUND(VAL: REAL): INTEGER;

Of the standard procedure identifiers, ZOOM Pascal 64 has implemented the READ, READLN, RESET, REWRITE, WRITE and WRITELN procedures. These procedures are discused more thoroughly in the following sections of this manual.

In addition to the above mentioned standard, predeclared identifiers, additional predeclard identifiers are available with ZOOM Pascal 64.

The predeclared type identifer STRING has already been discussed in the section of this manual on variable declarations. The string functions CONCAT, COPY, DELETE, LEFTSTR, MIDSTR, POS and RIGHTSTR are described in detail in a following section.

In addition to the above string functions, ZOOM Pascal 64 supplies six other non-standard functions. They are ANDB, NOTB, ORB, RND, GETKEY and INKEY.

The functions ANDB, NOTB and ORB allow Boolean operations on INTEGER expressions. The functions ANDB, and ORB require two parameters; NOTB arequires a single INTEGER parameter. All three return an INTEGER value.

The function RND requires a single parameter whose type must be either REAL or INTEGER. If the parameter value is positive, the function returns a new random number value. If the parameter value is negative, the function returns a random value after changing the seed value. In either case, the value is type REAL.

The functions **GETKEY** and **INKEY** allow **ZOOM Pascal 64** program to obtain the value of a keystroke on the system console. Both functions require no parameters and return a value of type **CHAR**. The function **GETKEY** returns a value of CHR(0), if no key is pressed. The function **INKEY** waits until a key is pressed.

In addition to the functions mentioned above, **ZOOM Pascal 64** supplies two non-standard procedures. They are **EXIT** and **CLOSE**.

The **EXIT** procedure may be used to exit from a procedure or function. The **EXIT** procedure requires a single parameter which is the identifier of the procedure or function in whose body the **EXIT** is located. To exit from the main body of the program, use the identifier **MAIN**. For examples of the use of the **EXIT** procedure, please refer to the modules **PMODVAL, PRANDCR** AND **PRANDUPD** which are on the distribution diskette.

The **CLOSE** procedure may be used to close a file opened via a **RESET** or **REWRITE** procedure. The **CLOSE** procedure requires a single parameter which is the identifier of the file to be closed. Use of this procedure is not required, since the run time package closes all opened files when a **ZOOM Pascal 64** machine language program ends. However, it must be used if you intend to access files on different diskettes inserted in the same disk drive during a course of a run.

3.6 STRING FUNCTIONS

ZOOM Pascal 64 includes all of the string functions supplied by UCSD Pascal. Additionally, **ZOOM Pascal 64** has the following differences:

* the string procedure **INSERT** is not implemented.
* two string functions available in Commodore BASIC are implemented: **LEFTSTR** and **RIGHTSTR**.
* the COPY function has been given an alias of **MIDSTR**.

Here's a description of the STRING functions:

1. **CONCAT(string1,string2,...,stringn)** - this function returns a string that is the combination of two or more string expressions into a single string. If any expression in the listof expressions are of type **CHAR**, an automatic conversion to type **STRING** takes place.

2. **COPY(string1,integer1,integer2)**
      or
   MIDSTR(string1,integer1,integer2) - this function returns a string containing the characters of the string1 beginning with the integer1-th position for a length of integer2 characters. It is equivalent to the **MID$** function in BASIC.
   e.g. COPY('THIS IS A STRING',6,5) returns 'IS A'

3. **DELETE(string1,integer1,integer2)** - this funcion returns a string containing the characters of string1 without the characters beginning with the integer1-th position for a length of integer2 characters.
   e.g. DELETE('THIS IS A STRING',6,5) returns 'THIS STRING'

4. **LEFTSTR(string1,integer1)** - this function returns a string containing the leftmost integer1 characters of string1. It is equivalent to the BASIC function **LEFT$**.
   e.g. LEFTSTR('THIS IS A STRING',4) returns 'THIS'

5. **LENGTH(string1)** - this function returns the an integer which is the length of string1. This function is equivalent to the BASIC function **LEN**.
   e.g. LENGTH('THIS IS A STRING') returns integer value 16

6. **POS(string1,string2)** - this function returns an integer representing the position of string1 within string2. If string1 occurs several times within string2, **POS** returns only the first occurance. If string1 does not occur

within string2, POS return a zero value. The POS function
accepts an expressionof type CHAR as string1, but not as
string2.
e.g. POS('A','THIS IS A STRING') returns integer value 9

7. RIGHTSTR(string1,integer1) - this function returns a
   string containing the rightmost integer1-st characters of
   string1. It is equivalent to the BASIC function RIGHT$.
   e.g. RIGHTSTR('THIS IS A STRING',6) returns 'STRING'

3.7 INPUT AND OUTPUT PROCEDURES

ZOOM **Pascal** provides six of the standard Pascal input and output procedures. They are:

* READ
* READLN
* RESET
* REWRITE
* WRITE
* WRITELN

The procedures READ and READLN allow the user to read data from the keyoard, a cassette file or a disk file, into a variable or a list of variables.

The procedures WRITE and WRITELN allow the user to write an expression or a list of expressions to the screen, a cassette file or a disk file.

The procedures RESET and REWRITE prepare the keyboard, screen, cassette file or disk file for data transfer.

If the first parameter of a READLN procedure is not a variable of type TEXT, then the cursor is positioned to a new line after filling the last variable in the parameter list. Otherwise, the READLN procedure is the same as the READ procedure. This means that the READLN procedure does not skip to the beginning of the next line when reading from a file.

Here's a sample program that gives you some examples of the READ procedure.

```
VAR CHVAR:   CHAR;
    INTVAR:   INTEGER;
    REALVAR: REAL;
BEGIN
    READ(CHVAR);          (* EXAMPLE 1 *)
    READ(INTVAR);         (* EXAMPLE 2 *)
    READ(REALVAR);        (* EXAMPLE 3 *)
    READ(INTVAR$);        (* EXAMPLE 4 *)
END.
```

Example 1 causes a single character to be read from the keyboard and stored in the variable called CHVAR.

Examples 2, 3, and 4 causes a combination of character input and data conversion. The number of characters read depends on the declared type of the variable being read into. For INTEGER and REAL variables, characters are read until a stopping character is encountered.

36

For INTEGER variables, a stopping character is any character other than the digits 0 through 9.

For REAL variables, a stopping character is any character other than the digits 0 through 9, the decimal point for the letter E.

For hexadecimal input (EXAMPLE 4), exactly four charcters are read.

This method of input is quite different from the way BASIC operatres. In BASIC, entering the string '39X' in response to a request for numeric input results in ?REDO FROM START. In ZOOM Pascal, entering the string '39X' in response to a request for numeric input does not result in an error message. A display of the variable read into shows a vlaue of 39.

The following program gives some examples of the **WRITE** procedure:

```
CONST   CHCONST   =  'A';
        STRCONST  =  'XYZ';
        INTCONST  =  13;
        REALCONST = 2.3E+4
BEGIN
    WRITE(CHCONST);
    WRITE(INTCONST-2),STRCONST);
    WRITE(1=1,REALCONST);
    WRITE(INTCONST$)
END.
```

The output of this program is:

        A11XYZTRUE 23000 000D

If all the **WRITE**s were changed to **WRITELN**, then the output is:

        A
        11XYZ
        TRUE 23000
        000D

If the first parameter of a **READ, READLN, WRITE** or **WRITELN** procedure is a variable of type **TEXT**, then the input/output operation involves a file instead of the keyboard or screen. Before attempting a **READ(LN)** or **WRITE(LN)** from or to a file, rather than the keyboard or screen, you must use a **RESET** or **REWRITE** procedure to establish a data path for that file.

The **RESET** procedure is used for input files and the **REWRITE** procedure is used for output files.
The first parameter of either a **RESET** or **REWRITE** procedure must be a variable identifier of type **TEXT**.

The second parameter is the external file name and may be either a string literal, a string constand or a string variable. The following program fragments show three different methods for opening the same disk file on drive 1 for input:

```
CONST   CONSTNAME = '1:ABC';
VAR     INFILE: TEXT;
        VARNAME: STRING[16];
        .
        .
        .
        RESET(INFILE,'1:ABC');
        .
        RESET(INFILE,CONSTNAME);
        .
        VARNAME := '1:ABC';
        RESET(INFILE,VARNAME);
```

A more complete discussion of external file access is covered in the following section.

3.8 EXTERNAL FILE ACCESS

Normally, a READ or READLN procedure reads from the keyboard
and a WRITE or WRITELN procedure writes to the screen.
However, input or output may be directed to an external file
if the first parameter of a READ, READLN, WRITE or WRITELN
is a variable defined as being of type TEXT. First you must
establish a data path using RESET or REWRITE.

The first parameter of a RESET or REWRITE procedure must be
a variable of type TEXT. This parameter is followed by a
variable number of parameters.

If the second parameter is an expression of type STRING,
then ZOOM Pascal considers it to be the file name. The
filename must be the last parameter. This two parameter
format (TEXT-type variable followed by STRING-type variable)
is referred to as the automatic format and can only be used
with sequential files on cassette or diskette. If a diskette
file is indicated, device 8 is used.

If the second parameter is not an expression of type STRING,
then it must be an expression of type INTEGER. ZOOM Pascal
64 uses this value as the IEEE device number. This second
parameter must be followed by a third parameter, also an
expression of type INTEGER and represents the IEEE secondary
address. The fourth parameter must be an expression of type
STRING but may be omitted if a file name is not appropriate
to the IEEE device being used. This three or four parameter
format is referred to as the manual format.

The automatic format is so designated because it uses the
command channel to check for a successful file open if the
file name prefix indicates a disk file. It also adds a
',S,R' or ',S,W' suffix to the file name if the file name
prefix indicates a disk file. For those who wish to read or
write 'PGM' files, then a ',P' suffix is allowable in the
file name and causes the ',S,R' or ',S,W' suffix normally
appended, to change to ',P,R' or ',P,W'.

The manual format does not add these suffixes. It does not
adjust the file name; it does not check for a successful
file open. If the device being used (e.g. drive number)
requires special characters in the file name, you must
provide them. Any special command channel programming must
be explicitly coded.

A READ or READLN can only be executed for a file opened with
RESET. A WRITE or WRITELN can only be executed for a file
opened with REWRITE. Therefore a random access file or
command channel requiring both reading and writing must be
accessed via two file definitions.

The distribution diskette contains sample programs on how to use files.

The source program **PCCEXMP** shows you how to read from and write to the disk command channel in order to scratch a file on a diskette.

The source program **PLSTCBMD** shows you how to access the directory of a diskette as a file.

The source program **PRANDCR** shows you how to create a random file on a diskette.

The source program **PRANDUPD** shows you how to perform a simple random file update on a diskette.

You may list these program or compile them and execute tl em.

3.9   OUTPUT FIELD WIDTHS

The **WRITE** and **WRITELN** procedures allow the use of field width specifications. A field width specification consists of a colon followed by an integer expression. The field width specification may be used with any data type.

Integer expressions may have a $ suffix to indicate that output in hexadecimal is desired. The standard width for hexadecimal output is four character positions. By using a $:2 specification, the field is a hexadecimal value two positions wide instead of four.

An example of a valid statement are:

        WRITE(1:2,1.1:4,'?':2,'ABC':4,TRUE:5,16$:2);

The output of the above statement is:

             1 1.1 ? ABC TRUE10
             ∬   ∬   ∬   ∬   ∬
    position 1...5...10...15...20

Padding spaces are added to the left of the output field, but only if the size of the output field is less than the width specification.

If the width specifications were omitted from the above example, the statement would be:

        WRITE(1,1.1,'?','ABC',TRUE,16$);

The output is then:

             11.1?ABCTRUE0010
             ∬   ∬   ∬   ∬   ∬
    position 1...5...10...15...20

3.10 ERROR MESSAGES

Most of the error messages displayed by the compiler are self-explanatory.

There is one message which is not. This message is **END TEXT FOUND.**

This message appears if there is no space or carriage return character following the last **END.** which terminates the Pascal program. This is acutally an eroneous error message.

This message also appears if you have an odd number of apostrophes in your source file or if you have a **(*** with out a matching *****).** Sometimes the only way to locate this problem is to direct the output of the compiler to the printer. If the P-code numbers get stuck on one value, then the problem is in the area of the last incrtement in the P-code number.

3.11 FILE NAME SYNTAX

If a null file name is entered, the system defaults to the cassette drive. If opening an input file, the first file found becomes the file read. If opening an output file, the fill is written without a name.

A unit designator is part of the filename when opening either a cassette input file, a cassette ouput file or a disk output file. A unit designator consists of a single character followed by a colon. For a disk file, the single character is either **0:** or **1:**; for a cassete file, the u'single character is a **T:**.

If a unit designator does not preceed the file name when opening an input file, the system assumes that a disk drive is to be used and searches the directoreis of both drives.

4.1 APPENDIX A - RESERVED WORDS

STANDARD PASCAL

AND	END	NIL	SET*
ARRAY	FILE*	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO*	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN*	PROGRAM	WHILE
DOWNTO	LABEL*	RECORD*	WITH*
ELSE	MOD	REPEAT	

ZOOM Pascal 64

CALL	MEM	SHL	SHR

* not implemented in ZOOM Pascal 64

4.2 APPENDIX B - PREDEFINED IDENTIFIERS

STANDARD PASCAL

CONSTANTS:
FALSE	TRUE	MAXINT

TYPES:
BOOLEAN	INTEGER	REAL	TEXT

FUNCTIONS:
ABS	EOLN*	PRED	SUCC
ARCTAN	EXP	ROUND*	TRUNC
CHR	LN	SIN	
COS	ODD	SQR	
EOF*	ORD	SQRT	

PROCEDURES:
GET*	PAGE*	READLN	UNPACK*
NEW*	PUT*	RESET	WRITE
PACK*	READ	REWRITE	WRITELN

FILES:
INPUT*	OUTPUT*

* not implemented in ZOOM Pascal 64

ZOOM Pascal 64

TYPE:
   STRING

FUNCTIONS:
ANDB	GETKEY	MIDSTR	RIGHTSTR
CONCAT	INKEY	NOTB	RND
COPY	LEFTSTR	ORB	
DELETE	LENGTH	POS	

PROCEDURES:
CLOSE	EXIT

4.3 APPENDIX C BIBLIOGRAPHY

Bowles, Kenneth L., MICROCOMPUTER PROBLEM SOLVING USING PASCAL, SpringerVerlag, New York 1977.

Conway, R., Gries, D., and Zimmerman, E., A PRIMER ON PASCAL, Winthrop Publishers, Cambridge, MA, 1976.

Grogono, Peter, PROGRAMMING IN PASCAL, AddisonWesley, 1978.

Kieburtz, Richard, STRUCTURED PROGRAMMING AND PROBLEM SOLVING WITH PASCAL, PrenticeHall, Englewood Cliffs, NJ, 1978.

Jensen, Kathleen and Wirth, Niclaus, PASCAL USER MANUAL AND REPORT, SpringerVerlag, New York, 1974.

Welsh, Jim and Elder, John, INTRODUCTION TO PASCAL, PrenticeHall, Englewood Cliffs, NJ, 1979.

Wirth, Niklaus, SYSTEMATIC PROGRAMMING: AN INTRODUCTION, PrenticeHall, Englewood Cliffs, NJ, 1973.

Wirth, Niklaus, ALGORITHMS + DATA STRUCTURES = PROGRAMS, PrenticeHall, Englewood Cliffs, NJ, 1976.

YOU CAN COUNT ON

Abacus
Software

P.O. Box 7211,   Grand Rapids, MI 49510   616/241-5510