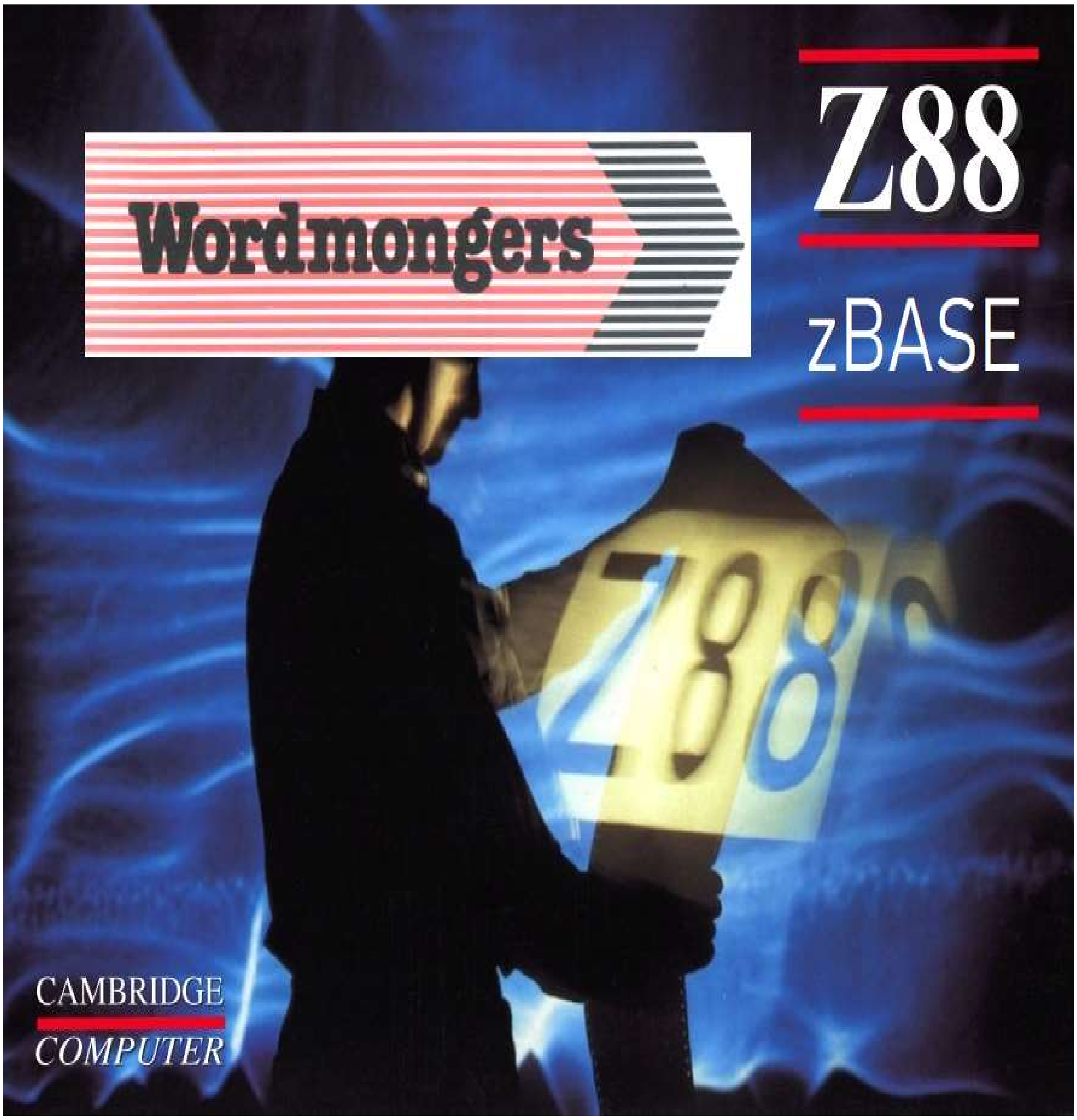
Wordmongers

Z88

zBASE



CAMBRIDGE COMPUTER

Congratulations on choosing zBASE. The Z88 Portable offers a wide range of built-in facilities. With the addition of zBASE, the Z88 comes of age.

zBASE operates in two modes. Interactively, data may be manipulated with a minimum of effort. For those who need specific input / output routines and screens, zBASE is also a programming language.

Complete application systems can now be written for the Z88 by people other than professional programmers.

Acknowledgement & Thanks

An essential part of the development of zBASE has been the participation of a large number of users in the testing of the beta version.

These brave souls risked all their Z88 data with a new and untested product. The first beta test version was subjected to some 400 hours of use and abuse with numerous undocumented features discovered in the process. Without the help of these testers zBASE could not have moved forward.

Their contribution assisted not only the production of the software, but also in many changes to the original draft of the manual.

Clive, Derek and Tony at Wordmongers would like to acknowledge the work done by the beta testers. Wordmongers accepts that the beta testers carry no responsibility for the final version of zBASE as published.

The beta testers were:

R.Beddard, Ian Braby, Mike Case, Tony Cox, G.C.Denney, John Dobson, R.C.Dorrance, Steve Drain, John Driver, Vic Gerhardi, C.M.Glover, S.P.Gray, F. W. Halliday, Gerald Hughes, B. P. James, Robin Jarvis, Charles Jenkins, N. A. Joseph, Dr Warren Kovach, Chris Lewis, Thomas Malinowski, M. Meijeraan, Francis Musgrave, M. Parker, Dr. L. Ratnasabapathy, C. M. Robinson, S.Fraas, K.G.Woolf, Roy Woodward, Matthew Soar, H.E.Shaw and John Hudson.

Content

zBASE Quick start guide

Getting familiar with a new piece of software is often tedious and sometimes daunting. To provide a Quick demonstration to those who are new to zBASE, this Quick start guide is a short tutorial aimed at providing a rapid introduction to te system.

For those familiar with the dBASE family of products, and for those experienced in writing programs in any language, this guide is intended as a summary of the introduction section of the manual. For such experienced programmers, the guide plus the reference section may be all that is needed. The initial sections of the Introduction sections should still be read especially concerning the warranty and disclaimer sections.

This guide is no substitute for reading the manual. Indeed, this guide will merely scratch the surface of zBASE capabilities. However, since zBASE needs liveware to make use of those capabilities, it is important to the authors that the liveware should feel some early reward for the effort of getting familiar with zBASE. Those rewards are plentiful in this guide.

With the INDEX on display on the Z88, open the clear perspex cover labelled '1 2 3', and insert the zBASE Application ROM in slot 2. Close the perspex cover.

The menu bar should be moved on the APPLICATIONS INDEX until it is highlighting the zBASE application.

Press ENTER to run zBASE.

When the } symbol, known as the curly prompt appears, type the word **QUIT** followed by ENTER. This action ends the use of zBASE. In version 1.2 you need to confirm the return to the APPLICATIONS INDEX, by pressing any key.

This is the only proper way to exit from zBASE. If the INDEX key is used, and the zBASE suspended application is KILLed, open database files will not be properly closed and permanent damage will occur to the data files.

The first real job is to create a data file. To do this, the file structure must be defined. This definition is done in PipeDream. Highlight the PipeDream application and press ENTER. This creates a new PipeDream suspended application.

Type in the following lines, exactly as shown.

COMP$,15
PHONE$,17
NAME$,20
KEYFIELD$,5

Now press ◇**FS**, followed by the file name, **PHBOOK.DEF**. Then move the cursor down 4 lines using the **ò**, to the question, 'Save plain text'. Enter a **Y** for YES and hit ENTER.

That has created the definition file for a phone book database. Now return to zBASE by pressing ◇**W**.

At the curly prompt (}), type in the command line
}CREATE PH.DBF FROM PHBOOK.DEF

When the } returns, the database file has been created and is open. Check its structure by typing

}DISPLAY STRUCTURE
The screen will appear as follows:

Database open in 1: is PH.DBF
COMP STRING 15
PHONE STRING 17
NAME STRING 20
KEYFIELD STRING 5
62 bytes/rec

Now create an INDEX file for easy searches.

At the curly prompt, enter the following
}**INDEX ON KEYFIELD$ TO PH.NDX**

That will establish an index file for use later.

Data input

After creating the file, the next step is to enter data. To begin with, the method shown will be the simple way of storing information. The below sets out two examples of how command files can make such work easier.

You are still in zBASE with the PH.DBF open, as confirmed by **DISP STRU** and **DISP STAT**. Type in the following..

```
}APPEND BLANK ENTER
}LET 1:COMP$="Wordmongers Ltd" ENTER
}LET 1:PHONE$="01296 - 43 78 78" ENTER
}LET 1:NAME$="Henry Webster" ENTER
}LET 1:KEYFIELD$="Wordm" ENTER
```

It is worth noting that the KEYFIELD is the field to be used for indexing purposes and rapid search capabilities. Therefore it can contain the first five letters of whichever field is to be used, e.g. when a company name is included, the first five letters from the company name could be used. Equally, when it's a personal contact with no company name, the first five letters of the name could be used, e.g.

```
}APPEND BLANK ENTER
}LET 1:COMP$=" " ENTER
}LET 1:PHONE$="020-7833-1212" ENTER
}LET 1:NAME$="Insp Bond" ENTER
}LET 1:KEYFIELD$="Bond" ENTER
}APPEND BLANK ENTER
}LET 1:COMP$="Short Brothers" ENTER
}LET 1:PHONE$="01494 885555" ENTER
}LET 1:NAME$="Morris Short" ENTER
}LET 1:KEYFIELD$="Short" ENTER
```

With at least a few records entered, a FIND process is required. This may be done directly at the curly prompt, or as for appending, by means of a command file as described in “Indirect variables”.

The zBASE command to use on an index file is '**FIND**'.

To find the entry for **Wordmongers**, at the curly prompt type

}**FIND "Wordm"** ENTER
With a small file, the next curly prompt will appear very quickly. As soon as it does, type

}**DISPLAY**ENTER

The record which has been found will be displayed. If a 0 is displayed it means that no match has been found. Try entering a shorter version of the key field e.g. **Wor**. If this finds say **Wortmongers** it reveals a typing error easily corrected by saying

}**LET 1:COMP$="Wordmongers"**
and do not forget the keyfield..

}**LET 1:KEYFIELD$="Wordm"**
If the next record is the one to be examined, type the command -

}**SKIP**
then at the next } type **DISP** to display the next record.

Note that only the first FOUR characters of any command need be used.

Looking for matches in an UNINDEXED file.

This can be done in a number of ways. '**LOCATE**' finds the first occurrence of the required match in a search from the beginning of the file, e.g.

}**LOCATE FOR 1:PHONE$="01"**

This command causes zBASE to start at the beginning of the file and search through the file, sequentially, checking each phone$ field to see if it starts with "01". When a match is found, the curly prompt returns. The **DISP** command will then display the record found.

As much of the field, or as little may be entered. The comparison always starts at the beginning of the field. The above locate command would stop at the first record starting with '01', whatever followed.

If 0 is displayed, no match has been found.

The second way to find records that match a particular criterion is to use the command **DISPLAY ALL**. This starts at the current record and then displays records that match the criterion set, 7 at a time. The operator must press any key to get the next 7 records, e.g.

}**DISPLAY FOR 1:PHONE$="01"**

While records are actually scrolling onto the screen, an ESC will abandon the search and return to the curly prompt. The record pointer will be at the record to which it was pointing when the ESC was pressed.

NB. ESC will not work at all if ESC =Off in v1.3 and when the 'Press SPACE to continue' prompt is on screen. A nifty piece of finger work of a SPACE followed quickly by an ESC will return control to the curly prompt in v1.2.

The **LIST** command is the same at the **DISPLAY** command except it simply scrolls on without stopping every seven lines. Holding down the SHIFT and t keys together will pause the scrolling when using the **LIST** command.

NB. The SHIFT key used must be the one on the left of the keyboard. (No we don't know why either.)

To close the file, enter
}**USE**

To re-open the file type
}**USE PH.DBF INDEX PH.NDX** ENTER

To check whether a file is open, type
}**DISP STATUS**

To check a file structure, enter
}**DISP STRUCTURE**

To return to the Z88 APPLICATIONS INDEX, ENTER
}**QUIT**.

NB. IT IS IMPORTANT THAT THE **QUIT** COMMAND IS USED AS THE METHOD OF CLOSING zBASE. If zBASE is KILLed as a suspended application, loss of data will occur because the files will not be properly closed.

An alternative method of entering data is to write a command file that makes it all a bit easier.

From zBASE, press INDEX leaving zBASE as a suspended application. From the Z88 APPLICATIONS INDEX, go into PipeDream by positioning the cursor over the PipeDream application and pressing ENTER.

Type or load the program labelled zBASE#PHENT.PRG. Do not use TAB characters. The lines that start with an ASTERISK (*) are comment lines that are ignored by zBASE. They do not have to be typed in at all.

After typing in PHENT.PRG it should now be saved. Press ◇**FS**, enter the filename **PHENT.PRG**, ⇩ 4 lines and enter **Y** to save the file as **PLAIN TEXT**.

Back in PipeDream, after the file save, enter □W to return to zBASE. When back at the } prompt, type the following

}**DO PHENT.PRG** ENTER

[If zBASE was QUIT rather than left suspended, the database file with its index must be re-opened before running PHENT.PRG. If this is the case, see QSG9 above for guidance on re-opening the file.]

Records may be added one by one as required using this command file. Note that the keyfield is automatically picking up the company name as the key, if a company name is entered. If not, the first five characters of the contact name are used. In either case, the offered key value need not be accepted. It may be overtyped with whatever content is required. If it is to be accepted, then press ENTER.

Input Rakewell's details.

- At Company enter **Rakewell**
- At the Name prompt, enter **Vic Gerhardi**
- When asked for Phone, type **01296-632491**

The Keyfield will display **Rakew**. Hit ENTER to accept it.

A few more records should be entered to give the database something to get its teeth into.

FINDing using a command file.

As before for PHENT.PRG, start a new PipeDream document and type or load in the program labelled FPROG.

Save this file using ◇**FS**. Use a filename of say **FPROG**. Remember it must be saved as **PLAIN TEXT**.

Return to zBASE either through the Z88 APPLICATIONS INDEX, or by simply pressing □ **W**.

With the phone book file in use, with its index file, at the curly prompt, type the following

}**DO FPROG**

Answer the prompt about the record to be found and the search will run. The cursor will remain at the next curly prompt after the FPROG command file has finished executing.

This quick start guide has been designed to provide new users with an opportunity to get something out of zBASE quickly. It does not reveal the flexibility available in zBASE nor does it demonstrates the wide potential for this database management program. The rest of the manual does go much further into the commands and functions contained in zBASE. Please take some time to review the rest of the manual and so enjoy the further capabilities of your Z88/zBASE combination.

zBASE Manual

The Quick Start Guide is designed to be a brief exposure to the major facilities of zBASE. Those new to database languages may find this a very useful starting point.

Section A of this manual, 'Introduction to zBASE', describes zBASE in descriptive terms with some examples of how the system can be used. Commands are dealt with in an order that are likely to appeal to a first time database user.

Section B - 'zBASE Reference', is an alphabetical list of the zBASE commands, with the syntax for each one dealt with separately. Following that, the zBASE functions are dealt with in similar fashion.

Section C - 'Sample Programs and Glossary', contains some sample programs that can be written in the zBASE language with a brief Glossary that explains some of the terms used in the manual. Also shown is a series of zBASE programs that together form a Stock Check system,

Introduction

The Wordmongers zBASE suite is designed to provide general database management opportunities for the expanded Z88. The Z88 must have at least a 128K RAM expansion cartridge in slot 1.

In the interactive mode, commands typed at the } prompt will be executed immediately.

Alternately, regularly used sets of commands may be put together into a command file to save repeated entering of the same commands. Those familiar with the dBASE family of programs from Ashton-Tate will find that there are some marked similarities between zBASE and dBASE II. (See trademark notice)

Copyright & Trademark notices

See Copyright & Trademark notices

Disclaimer

See Disclaimer

Handling, ROM's & installing the software

zBASE will only run on an expanded Z88. This means that the Z88 must have at least a 128K RAM cartridge installed. If a single RAM is fitted, it must be in slot 1.

***See* “** Fitting & Using the ROM”.

After installing the ROM, zBASE will appear on the Z88 INDEX with a □**W** code.

To run zBASE, move the cursor to the zBASE application option in the INDEX and hit ENTER, or ENTER □**W**.

Important basic concepts

zBASE is a command driven database language with over 40 commands and functions. This provides a powerful facility for programming the Z88 for data manipulation. To those familiar with dBASE II, the granddaddy database program, there should be a feeling of having seen it all before. Given that imitation is the most sincere form of flattery, these similarities are purely intentional. However, for a variety of reasons, not least of which is a 32K ROM space, not all dBASE II facilities are emulated. Equally, some have been altered in an attempt to improve and to provide a better fit with the Z88.

Certain elements of the zBASE system need to be explained early on. These elements relate to the use of variables and files.

Database areas

Data files may be opened in either of two areas. These areas are called PRIMARY and SECONDARY. These areas are labelled as 1: and 2: respectively.

The 1: and 2: symbols are referred to elsewhere in this manual as the usage area. Whenever a field variable is addressed, it must be prefixed with this usage area symbol. The single exception to this is in the use of a field's list in the LIST and DISPLAY commands.

Variables

Variables are either memory variables or field variables. A memory variable is a label for a pigeon hole containing a number or a string of letters. Such memory variables or MVARS are volatile. They exist only for as long as the zBASE program is in use. When zBASE is QUIT, memory variables are lost. When zBASE is left as a suspended application, memory variables remain preserved along with the rest of the program.

Field variables or FVARS, relate to data in a data file. Such variables always begin with the usage area symbol. (See 1.1 above.) e.g. 1:fieldname

Field variables and memory variables are CASE INSENSITIVE. Upper and lower case names using the same characters will be regarded as identical.

Variable names can have up to 8 characters.

A variable that contains a string of letters must always end with the string symbol, namely a dollar sign, $. A numeric memory variable needs no such suffix.

Accordingly, a memory variable used for temporary storage of a name might be called, **msurnam$**. The equivalent field variable would be called **1:surnam$** or **2:surnam$**.

Before a memory variable may be used, it must be initialised. This is done by storing a value to that memory variable using the **LET** command.

The limit on total memory variables in use at any one time is 512 bytes. To control this use the RELEASE command.

Memory variables example:

```
}LET user$="FRED" ENTER
}? user$ ENTER
FRED
}LET number=42
}? number ENTER
42
}? number * 12 ENTER
504
}LET NEWUSER$="Sheila" ENTER
}? user$+" and "+newuser$ ENTER
FRED and Sheila
}
```

Allocating a value to a variable is called assigning. To assign a string to a variable, the value assigned must be enclosed in quotation marks. See user$ above.

To assign a numeric variable, the variable name must not end in a $ symbol and the value must not be enclosed in quotations.

The result of a function may also be assigned to a variable, e.g. to set up a variable holding the current system date:

```
}LET today$=date() ENTER
}? today$ ENTER
08/08/88
}
```

```
}LET ramspace=ram() ENTER
}? ramspace ENTER
17408
}
```

In assigning function results to a variable, care must be exercised in ensuring that the variable name used matches the result of the function. DATE() returns a string whereas RAM() returns a number. The variable names used reflect that. A **DATA TYPE MIS-MATCH** error will result from ignoring this rule.

Field variables are treated just like memory variables except they have a 1: or 2: in front of their name to distinguish them.

Interactive versus Command file

zBASE operates in two different modes. One is called the interactive mode and the other is command file driven.

The interactive mode is the mode in which zBASE starts up when first invoked. After the copyright message, a curly bracket will appear. This is referred to as the curly prompt and represents the interactive mode. Commands and functions are entered at this prompt and executed immediately.

In command file mode, a set of zBASE commands are put together in a PipeDream plain text file. At the curly prompt, the command DO FRED will call the file called FRED and will execute the commands found in that file.

When a series of commands are used often, in sequence, it is useful to avoid entering them at the curly prompt every time they are needed. Instead they may be recorded to a command file and executed as a set when required.

Some commands and functions will only operate in a command file. If they are used at the curly prompt the error message

COMMAND FILE ONLY

will appear.

If it is present, the file **ZBRUN** will be run automatically when zBASE is started. It acts like a BOOT.CLI file.

An introduction to databases

The following sections describe the building of a database file and its interrogation. It should be read through to the end. However, computer users are notorious for skipping manual pages. In this case, all the available commands with their syntax are explained in Section B and those wishing to learn by their mistakes are invited to use that section of the manual.

In this Introduction section, an example will be used to demonstrate techniques and commands. The system is a video tape cataloguing system. It is designed to keep track of a domestic video tape library in a fashion similar to that used for books. The specific command programs used can be found in - zBASE Programs.

Files

A database file is a collection of records. Each record contains details of 1 item, e.g. a library database file would contain separate records for each book in the library. A phone book might have one record per person recorded. In a Video Library system, each record would represent a single programme or film.

Only alpha-numeric characters should be used in file names. These are A-Z and 0-9. Other characters are regarded by OZ, the Z88 operating system, as file name terminators. Therefore OZ will not differentiate between files called MENU and MENU* or MENU#.

Many database systems assign file name extensions that show what type of file it is, i.e. file.DBF shows that the file is a DATABASE File. file.NDX would be an INDEX file. This is only a convention as far as zBASE is concerned and does not need to be regarded as a rule. Having said that, some form of file name convention is useful and the sample system for Stock Control, contained in Section Cof the manual does name all command file programs with an extension of .PRG.

zBASE database files contain certain information at the top of the file that is not readable by PipeDream. Therefore zBASE data files should not be opened under PipeDream. If this does occur then the file should not, under any circumstances, be SAVED from PipeDream. It should be KILLed. If it is saved from PipeDream, the header information will be destroyed.

Records

The limit on the number of records in a database in zBASE is 65535 records.

Each record in a database consists of different fields of information.

Fields

In designing a database, much of the effort is in getting the field structure correct for the required application. The classic consideration is whether to split a name into surname and forename or to hold the whole name as a single field.

If the data are to be used only for addressing envelopes, then holding the name as a single field will be acceptable. However, if the data are to be listed in alphabetical order of surname, then the parts of the name must be split into separate fields within the record. It is still part of the same record, but the design of the record is changed.

A field of information may be thought of a single box of information within an outer box. The outer box is the record and each box it contains is a field. In a phone book example, the SURNAME$ field is obviously different from the FORENAME$ field. The separation is made because searches are likely to be required on the SURNAME$ field and because when reports are obtained, the SURNAME$ field is to be easily identifiable. Also, there is a strong possibility that the file will have to be sorted on this field to get an alphabetical listing.

If the name was in a single field into which both these bits of information were stored, then sorting would be an interesting task, viz

File with name as two fields:

Forename$, Surname$
Bloggs Freda
Jones Hannibal
Assuah-Kwesi Rick
Kuczynski Irving
Alexandrou Francesca

File with name as single field Name

Freda Bloggs
Hannibal Jones
Rick Assuah-Kwesi
Irving Kuczynski
Francesca Alexandrou

On sorting these two files the problem becomes apparent in that sorting on SURNAME$ is the usual way of doing things, whereas sorting on the whole name, starting with forename is much more unlikely. If the file is to be sorted on name, that usually means surname. If that is the case, then SURNAME$ must be identifiable as a single field.

Although this appears elementary in the case of name fields, it is nevertheless an important consideration whatever is being stored in the file.

Data types

There are two types of data, strings and numbers. Strings are simply collections of letters and or numbers that are to be regarded as TEXT. Numbers are numeric values upon which some form of calculation may be required.

Users familiar with dBASE will be aware of a further data type, namely logicals. In zBASE, logicals may be simulated on the basis described here.

LOGICALS are really of type numeric. If the value is 0, the answer is taken to be FALSE. Any non-zero number is regarded as TRUE. In zBASE, a field to be used as a logical must be initialised as a NUMERIC field.

Very few items in a database are of type number. Again referring to a Video Library system, the film reference number (REF_NUM$) may be numbers. However, they will not be used for calculation purposes so they may be viewed as STRINGS. The rule is that when numbers are used as labels for an item, such as part numbers or code numbers, even if they are made up of numbers only, the field is still a string field.

Similarly, a phone number is made up solely of digits but because they are not involved in calculations they are usually held as STRING fields.

Index key fields

Strictly speaking this is not a different data type. However, it is a significant consideration when designing databases so is noted here as a consideration to be included in such a design.

An index key in zBASE is the field in the record that is to be used for finding data records quickly. It is also the field to be used for the sorting of files. Having said that, indexing a file does not actually sort it. The file simply appears to be in the new order for all purposes.

An index key field can be any string field. If the file to be indexed is expected to be large, the index field should be kept as small as possible. This then means that the index file created is as small as possible and the time taken for a rapid search is also as small as possible.

How to create a data file

Having worked out the fields required in the database, the process of creating that file under zBASE starts with PipeDream.

The elements of creating a zBASE database file are as follows.

Firstly, a PipeDream document is created defining the fields and their widths. This document is saved as PLAIN TEXT.

From within zBASE itself, the CREATE command is then invoked. Each step is described in more detail below.

Defining fields, width, type

From the Z88 INDEX, use the cursor to access PipeDream by moving the cursor to the PipeDream application and pressing ENTER. A new PipeDream document will be opened.

With the cell number A1 displayed at the top left hand corner of the screen, type the following:

```
REF_NUM$,3
TITLE$,20
TYPE$,3
TAPE_NO$,3
DURATION$,5
RATING$,1
```

The pattern to note for defining file structures is explained in more detail in Section B - zBASE Reference. In short, each line contains details of an individual field. If the field is a string field the name must end with a $, it must be followed by a comma and must end with a number defining the width of the field. A numeric field is simply described with its name and no $ and no width.

Having defined the file structure, it must be saved. Press ◇**FS**, followed by a file name, say **VIDEO.DEF**. Move the **ò** 4 lines to the Save plain text prompt and enter <Y>. The ENTER key will then save this structure file in a form to be accessed by zBASE in the next stage.

Creating the file within zBASE

Returning to zBASE, the next stage occurs at the curly prompt.

The command to be used is CREATE FROM. The syntax is

CREATE <database filename> **FROM** <PipeDream define filename>

e.g. if the define file is called VIDEO.DEF, and the name wanted for the database file is VIDEO, the command line would be:

}**CREATE VIDEO FROM VIDEO.DEF** ENTER

The file is then in an open state at record 0 in whichever usage area was active at the time the command was entered.

If there was a file open, it will be closed automatically by the CREATE command.

If data is ready for input, the file should be left open. To store data to the file, the fields are treated in a fashion very similar to memory variables. The only difference is that field variables from files are preceded by a **1:** or **2:**. (See Section 1 above or Section B later.)

The choice of where a file is opened is managed by the **SELECT** command. (See INTRO part 17 and Section B of the manual.) For the moment, assuming the **SELECT** command has not been used, the assumed area of operation is PRIMARY, labelled 1:.

To put in some data, each field is filled separately. To begin with, say the first video to be entered is 'Flight of the Condor'. The reference number is to be "001". The type code is to be 'THR' for thriller, the tape number is "101", duration is 2.25 hrs. and the rating is B.

In order to get a new record open, ready for this information, at the prompt enter:

```
}APPEND BLANK ENTER
```

This command creates a new record with all fields blank. Numeric fields are set to 0 (zero).

At the next } prompt, use the LET command, similar to the BASIC LET command, with the database area as a prefix for the field name viz:

```
}LET 1:REF_NUM$="001"
}LET 1:TITLE$="Flight of the Condor"
}LET 1:TYPE$="THR"
}LET 1:TAPE_NO$="101"
}LET 1:DURATION$="2.25"
}LET 1:RATING$="B"
```

The **DISPLAY** command would now reveal the whole record and will look something like this.

```
}DISPLAY
1 001 Flight of the Condor THR 101 2.25 B
}
```

Further **APPEND BLANK** commands may be used to add more records.

Opening and closing database files

To open a database file, the command word is **USE**. e.g.

```
}USE VIDEO
```

If a file was open in that usage area at the time a USE command is executed, that previous will be closed.

When entered on its own, without a file name, e.g.

```
}USE
```

the file currently open will be closed. No new file will be opened.

DISPLAY STAT will show that there is no file open.

```
}DISP STAT
1 is active
```

That means that the current database area is area 1. See using multiple databases for further information about database areas.

When entered with a filename, the selected file will be opened and the first record in the file will be available immediately.

```
}USE VIDEO
}DISPLAY STATUS
1: VIDEO
Recs/file 1 Current rec 1 Bytes/rec 42
1 is active
```

This tells you that the file open is VIDEO. Total records on the file is 1. The current record is number 1 and that there are 42 bytes in each record. In v1.3 there is an additional line to show if **-ESC-** is ON or OFF.

The full syntax of this command is shown below:

USE [FILENAME]

The file will be opened in the currently selected area. If in doubt about the current area, enter the command

```
}DISPLAY STATUS
```

When a file is initially opened, the current record will be the first one in the file. The record number will be 1. If the file is an empty file, the record number will be 0 and the End of file function, **EOF()** will be true, i.e. 1 or some other non-zero number.

To check the 'End of file' function type,

```
}? EOF()
0
```

Remember that ZERO means FALSE. Any other value, usually 1, means TRUE.

Apart from the **USE** command, the **QUIT** command will also close all files before going back to the Z88 applications menu.

Checking structure

A command is available to display the structure of the currently open file. The command is

```
}DISPLAY STRUCTURE
Database open in 1: is VIDEO
REF_NUM STRING 3
TITLE STRING 20
TYPE STRING 3
TAPE_NO STRING 3
DURATION STRING 5
RATING STRING 1
42 bytes/rec
```

Indexing and Index files

Most databases contain information which is entered in a random fashion. There is no pattern to the way the data are organised. zBASE uses a system known as INDEXING as a means of holding a file so that it appears to be in a specific sorted order.

This is done by the creation of a second file, used in parallel with the database itself. This parallel file, or INDEX file as it is called here, holds only a small part of the database information. Specifically it contains the data from the selected field to be used as the KEY, plus the position in the database file where the full record is to be found.

e.g. if the VIDEO file were indexed on REF_NUM$, the index file record would hold the code field, say "001", the record number "1", plus a few bytes more used for housekeeping. Since the index file is much smaller than the main database file, searches can be carried much more quickly. When the required record is found in the index, zBASE can go to the full record in the main database very quickly too.

[On a database of 580 records from a phonebook, containing name, company, phone and a 5 character key field, the typical time for a search on that keyfield is 1 second.]

In the Stock control system contained in Section C, the usual method of finding a product is by its code. Therefore, the file will be indexed on the CODE$ field.

To create the index file

some space must be freed in order to minimise the total memory used by zBASE. In this context, the space used is the database area 2.

Therefore, the start of the process is to ensure that any files open in area 2 are closed, as follows

```
}SELECT 2
}USE
}SELECT 1
}USE
}DISPLAY STATUS
1 is active
```

This tells you that the currently selected database area is number 1 but that no files are open.

The ground is now set for creating an index file.

First the database file is opened. Then the index command is entered specifying the key field name and the name to be used for the index file.

The following lines should be entered as shown.

```
}USE VIDEO
}INDEX ON REF_NUM$ TO VIDEO.NDX
```

That's all.

When further data is entered, any product will be retrievable by the use of the FIND command on entering the REF_NUM$ for the wanted item.

Entering **USE** will close the database file and the index file. To re-open the two files together enter:

```
}USE VIDEO INDEX VIDEO.NDX
```

The general form of the command line is

USE FILE
INDEX ON KEYFIELD TO INDEXFILE

With an INDEX file open, the database file will appear to be in the order of the keyfield, irrespective of case. i.e. lower case letters will be treated as upper case letters. If this were not the case, all items starting with lower case a's would follow those starting with upper case a's.

The LIST and DISPLAY commands will both access the INDEX file for the order in which the display is to take place.

Using FIND

Use of the FIND command with an INDEXed file is the fastest way to search for a specific record. **FIND**ing a record rapidly is achieved with an INDEX file as follows.

Say the item to be found was in VIDEO with a REF_NUM$ of '001'. It could be found by either assigning the sought value to a memory variable, then doing a find on that, or directly at the curly prompt by entering the sought value itself. e.g.

```
}FIND "001"
}DISPLAY
1 001 Flight of the Condor THR 101 2.25 B
```

Equally, if used in a command file, the sought value would be assigned to a memory variable such as mseek$.

```
}LET mseek$="001"
}FIND mseek$
}DISPLAY
1 001 Flight of the Condor THR 101 2.25 B
```

If the display only shows a 0 (zero) character, that means that no match has been found. EOF() will return a 1 (TRUE).

In command files, whenever a FIND is carried out, the next line should test for EOF()=0. If it is 0 (FALSE), then a match has been found.

Adding data to an indexed file

The fact that a file is indexed makes no difference to the adding of data except that a time allowance must be made for the updating of the index file.

As long as the INDEX file is open when the APPEND BLANK command is entered, the INDEX file will be properly updated.

Equally, if an INDEX file is open and an APPEND FROM command is used, the INDEX file be properly updated.

Tech note re INDEX file sizes

Given the Z88 environment, it is imperative that the amount of memory grabbed by zBASE should be kept to a minimum. This is to reflect the Z88 use of memory for all its functions and the ability to leave zBASE as a suspended application. If this is to be possible, it must use only a minimum of memory.

Given this constraint, it was decided that the indexing function would be allowed access to a 5K bytes buffer. In this context, a 5K buffer would allow the indexing of a keyfield of 5 characters on a file of about 700 records without resorting to paging and swapping of memory.

It also means that a file of say 250 records could be indexed very quickly even with a keyfield of say 20 characters.

Retrieving data

Apart from the FIND method described above, there is another way of finding particular pieces of information. This is by the use of a zBASE command called LOCATE which can search on any field. This is compared with FIND which only operates on the designated key field.

The LOCATE command examines the database for a given condition. In the VIDEO example, the search might be for all those programmes which are on a specific tape. The command line looks like this.

}**LOCATE FOR 1:TAPE_NO$="101"**

When the curly prompt returns, a **DISPLAY** will display the found record. If all that is displayed is a 0, it means that a match has not been found.

If the search is to continue for the next hit, the command to be used is

}**CONTINUE**

This literally continues the search down the file from the point of the most recent hit.

Another retrieval method is to use the **LIST FOR** command. For the example above the whole line would look like this..

}**LIST FOR 1:TAPE_NO$="101"**

This will list each record for which the condition evaluates to TRUE from the current record to the end of the file. Note that if the file is indexed, the listing will appear in keyfield order.

A further option with this command involves specifying the fields to be displayed for each hit. If the only fields wanted were say TITLE$ and REF_NUM$, the command line would be entered as..

}**LIST FOR 1:TAPE_NO$="101" FIELDS TITLE$, REF_NUM$**

The LIST command will scroll the hits one after the other. To get the hits to stop automatically after 7 records have been displayed, the **DISPLAY** command should be used. The syntax is exactly the same as for LIST.

}**DISPLAY FOR 1:TAPE_NO$="101" FIELDS TITLE$, REF_NUM$**

When used in conjunction with an indexed file, the DISP and LIST commands are very powerful. The FIND command is used to arrive at the first occurrence of the required value, then the DISP or LIST command, with its condition, can be used to show all the records which match, from that point in the file to the end.

Entering data

APPEND BLANK

This command literally means add a blank record to the database file. It creates an empty record at the bottom of the database file in use in the currently selected area. Each field starts empty. Numeric fields start with a 0 (zero).

In order to enter data to the record, each field must be dealt with independently. Values are assigned to a field using the **LET** command. To record a new VIDEO, the entry routine would be as follows:-

```
}USE VIDEO
}APPEND BLANK
}LET 1:REF_NUM$ = "100"
}LET 1:TITLE$ = "Bytes of Affection"
}LET 1:TYPE$ = "COM"
}LET 1:TAPE_NO$ = "012"
}LET 1:DURATION$ = "1.15"
}LET 1:RATING$ = "A"
```

Another APPEND BLANK would immediately open another blank record ready for the next item.

Variables and Top Bit Characters

It is possible to have any name stored in a field of a zBASE record by using the decimal code of a character: For example:

```
LET 1:LSTNAME="V"+CHR(228)+"is"
+CHR(228)+"nen"
```

The field LSTNAME will then contain the name Väisänen. On a British home market Z88 screen the foreign letters will be represented by black squares, but will print out on a printer tuned to accept the ISO character set (standard on most).

Alternative Method

An alternative method is to use a program called 'APPEND.PRG'. This routine emulates the dBASE APPEND command. The database to which data are to be amended must be open in area 1. TheAPPEND.PRG program must have been typed in under PipeDream and saved as PLAIN TEXT. At the } prompt, enter

}**DO APPEND.PRG**

A series of prompts will appear asking for the data to be entered for each field. The field name will appear next to each prompt.

After all the fields are filled, a further prompt will appear asking for a Y/N response as to whether there are more records to be entered.

It is worth noting that as soon as this program is called, a blank record will be created on the file open in area 1.

For those who have no wish to get involved with zBASE program writing, the structure of APPEND.PRG is of no importance. It is offered as a short hand way of easily entering data to any database. See the introduction to Section C for details of how to obtain this and the other programs on a Z88 EPROM. download or on a floppy disk.

Using command (do) file

Within a command file, the APPEND BLANK command is still the only way to get a new record added to a file. However, a complete input screen may be devised using the AT SAY GET commands. This system provides the opportunity to have an input screen with proper prompts which automatically puts the data into the file without the repeated use of LET.

For any repeated use of data, a command file is the answer. As with all programming, it takes longer to get the initial instructions into the system, but the speed of subsequent operations is greatly enhanced.

The next section is a blow by blow analysis of the input routine called, VIDINP.PRG.

```
* VIDINP.PRG V1.00 By Derek Fountain
CLS
SELE 1
USE VIDEO
```

This first section clears the screen and sets up the PRIMARY USE AREA. This is part of the command file just in case the currently selected area is secondary.

Having selected the appropriate area, the chosen file is opened.

```
GO BOTTOM
LET nref_num=VAL(1:REF_NUM$)
```

In order to establish the last reference number used, the bottom of the file is checked for the last used reference number. This is stored, as a numeric memory variable, under the label nref_num.

```
DO WHILE 1=1
```

This is the beginning of a DO WHILE loop. As long as the condition evaluates to TRUE, i.e. as long as 1=1 evaluates to a non-zero value, the command lines between the DO WHILE and the ENDDO will be executed repeatedly.

In this case the condition will always be true and is said to be an infinite loop.

As the processor hits the ENDDO, the condition for continuing is re-evaluated. If it is TRUE, the process is repeated. If it is FALSE, the program will drop out of the bottom of the loop and will resume running at the first command line after the ENDDO.

In the case of an infinite loop, the way is by using the RETURN command. See later.

```
AT 0,28 SAY "Home video library system"
AT 2,51 SAY "Tape :"
AT 4,16 SAY "Title :"
AT 4,51 SAY "Type :"
AT 5,16 SAY "Length:"
AT 5,51 SAY "Rating:"
```

To encourage accuracy in computer input, messages to operators should be clear and simple. Even if the programmer is herself to be the operator, later use of a program will always be easier if the screen messages are clear.

The AT SAY GET commands are for screen formatting. It places the screen at the disposal of the programmer.

The first number is the line number. Numbering starts at the top of the screen at line 0 and goes down to line 7. The second number is the character number starting at 0 on the left of the screen and going to 79 at the right.

These AT SAY commands display the field names to act as the prompts for the input.

```
LET nref_num=nref_num+1
```

```
LET mref_num$="&nref_num"
```

Having collected the last reference number used and stored it as nref_num, these two lines of code add 1 to the value, then translate the new value into a string variable ready for putting into the next new record.

```
LET mtape_no$=" "
LET mtitle$=" "
LET mtype$=" "
LET mduration$=" "
LET mrating$=" "
```

This section is about initialising memory variables. Any variables which are to be used must be created first. The names selected are meaningful in that they are to match the fields in the database file.

Additional variables are set up as they are needed. See confirm$ below.

```
AT 2,16 SAY "Ref number: "+mref_num$
AT 2,58 SAY mtape_no$
AT 4,23 SAY mtitle$
AT 4,58 SAY mtype$
AT 5,23 SAY mduration$
AT 5,58 SAY mrating$
AT 7,23 SAY "Enter the information on the video"
```

This set of instructions is about setting up the screen with the new reference number, at line 2 column 16. This command line displays the prompt contained within quotation marks and then follows that title with the newly assigned reference number.

The next series of AT SAY commands display the memory variables just set to spaces in the section above. This serves the purpose of over-writing any previous data that was left in those positions on the screen.

```
AT 2,58 GET mtape_no$
IF mtape_no$=" "
RELEASE nref_num,mref_num$,mtape_no$,mtitle$,mtype$,mduration$
RELEASE mrating$,confirm$
RETURN
ENDIF
```

The AT GET command is for operator input. The program stops with the cursor sitting in the position defined by the AT GET and waits for input from the keyboard. The input must be followed by ENTER . In this example, the input from the operator will be stored in the memory variable called mtape_no$.

IF/ENDIF pairs test for particular conditions and act accordingly. In this case the operator response is compared with a set of blanks. If the input was blank the IF statement evaluates to TRUE and the contents of the IF/ENDIF are executed. When the IF statement evaluates to FALSE, i.e. something other than blanks was entered to the mtape_no$ variable, the contents of the IF/ENDIF are ignored and the program continues running at the first line after the ENDIF statement.

By entering blanks, or rather by pressing ENTER with the variable blank, the program releases the currently held memory variables and then RETURNS to the calling program. If the program was called from the curly prompt, control reverts to the interactive mode.

The indentations are not part of the command file syntax. Indenting the contents of an IF/ENDIF branch makes it easier to read and identify errors. With this pattern, each IF aligns with its corresponding ENDIF. This provides an opportunity to check that for each IF there is a corresponding ENDIF.

```
AT 4,23 GET mtitle$
AT 4,58 GET mtype$
AT 5,23 GET mduration$
AT 5,58 GET mrating$
```

This small part of the program is what it all started with. This is the bit that actually gets the data from the operator and stores it in memory variables ready for putting into the data file later.

```
LET confirm$=" "
DO WHILE confirm$=" "
AT 7,23 SAY "Please confirm this information: "
AT 7,56 GET confirm$
IF WHERE(confirm$,"YNyn")=0
LET confirm$=" "
ENDIF
ENDDO
```

This routine is a very common section of any database input program. It provides the opportunity for the operator to check the input before actually appending the data to the file.

This section starts by assigning a single blank space to the variable called confirm$.

The DO WHILE loop will execute as long as confirm$ is a blank space. If it contains anything other than a blank space it will not execute. Therefore it can be seen that this DO WHILE loop will always run at least once because the confirm$ variable is set to a blank space immediately before the DO WHILE is invoked.

On the first time into the loop, the prompt is displayed asking for confirmation that the information is correct. The operator is then expected to enter a Y or an N.

Immediately after the operator has responded, the program tests the input to see if the character input is one of "YNyn".

The use of the **WHERE** command should be noted as a very useful routine for checking data upon entry. If confirm$ is NOT one of the four characters listed, the IF statement will be TRUE and confirm$ will be reset to a blank. If confirm$ is one of "YNyn", then confirm$ will be left as it was input.

In the end, the program reaches the ENDDO with a value for confirm$. The value will be one of "YNyn" or it will be a blank space.

Having hit the ENDDO, the DO WHILE line is re-evaluated. If confirm$ is a blank space, the DO WHILE line will still evaluate to TRUE, therefore the contents of the DO WHILE loop will be executed once more.

Alternatively, if the operator has input an acceptable character, the DO WHILE is finished and confirm$ contains a valid letter.

Execution of the program, continues with confirm$ being one of "YNyn" and nothing else.

```
IF UPPER(confirm$)="Y"
APPEND BLANK
LET 1:REF_NUM$=mref_num$
LET 1:TAPE_NO$=mtape_no$
LET 1:TITLE$=mtitle$
LET 1:TYPE$=mtype$
LET 1:DURATION$=mduration$
LET 1:RATING$=mrating$
* FLAG A
ELSE
LET nref_num=nref_num-1
ENDIF
```

confirm$ is first checked to see if it is a Y or a y. The upper() function means that if it was entered as a lower case y it will be converted to upper case before evaluating the If condition.

If upper(confirm$) was entered as 'Y', then the contents of the IF/ENDIF are run i.e. a new record is added to the database file and the contents of the new record replaced with the required values as collected in the memory variables above.

If the answer to the confirm question was N or n, the IF is not satisfied and the program drops through to the ELSE alternative. In this case the ELSE option reduces the reference number by 1 because it has not been used. When the program loops around again the 1 will be added back for the next record. If it were not reduced here, it would have clocked up 2 before actually going on a record.

The '* FLAG A' command line may be safely ignored at this stage. In the section below dealing with the use of multiple databases, this line is changed to something else to provide for additional related input.

```
ENDDO
RETURN
```

The final part of the program is the test to see if DO WHILE loop is to be repeated. As the program hits the ENDDO, the DO WHILE condition is re-evaluated. If the condition is met, the DO WHILE loop operates. If the condition is not met, the program proceeds back down to the ENDDO and on to the line after the ENDDO where the next command is executed.

In this case, the condition tested is whether 1=1. The test is that the DO WHILE loop will operate when the condition following the DO WHILE command evaluates to TRUE, or non-zero. Since 1=1 is always TRUE, the loop will continue. The only way out is to enter a blank tape number at the first input opportunity.

The RETURN command confirms the step back to the interactive mode at the curly prompt. When multiple command files are in use, the RETURN command will return the program to the previous level of command file.

With this program there is no message to the operator saying how to exit from the routine. As a first step in programming it might be a useful exercise to introduce a message saying 'Leave tape number blank to exit'. Alternatively, a different DO WHILE loop might be used to that a prompt could appear at the bottom of the program asking if there were more videos to be entered. If the response were 'N', the DO WHILE could be made to fail and so return control to the calling program or the curly prompt.

Amending data (Changing values, deleting records)

At curly prompt

Amending data is the same operation as entering data. The same **LET** command is used and the new data overwrites the old.

Having found the record which is to be amended, each field in that record may be amended simply by re-assigning a new value. It is as if the field were being re-defined, which of course it is.

Say a FIELD variable were set up as

```
}LET 1:TYPE$="COM"
```

This can be amended to 'THR' by simply repeating the line

```
}LET 1:TYPE$="THR"
```

It is often safer to check the value of a field before it is replaced with another value so a possible scenario might be:

```
}? 1:TYPE$
COM
}LET 1:TYPE$="THR"
}? 1:TYPE$
THR
```

If the new value is shorter than the previous value, spaces must be used to over-type the remaining bits of the field contents if a **GET** is used. e.g.

```
}? 1:TITLE$
}Bytes of Affection
}LET 1:TITLE$="Dracula in Love"
}? 1:TITLE$
Dracula in Love
```

If this same exercise were carried out using the **AT GET** system it would appear as follows

First change the field back to "Bytes of Affection"

}**LET** 1:TITLE$="Bytes of Affection"

Then clear the screen ready for the new exercise.

}**CLS**
}**AT 2,50 GET 1:TITLE$**
Bytes of Affection

Now overtype the new title wanted..

and it will appear as follows .....
Dracula in Loveion
with the cursor flashing on the 'i' after Love.

If ENTER is pressed at this point, the field will contain the characters as on the screen.
Press ENTER and then

}**?** 1:TITLE$
Dracula in Loveion

This time type:

}**AT 6,50 GET 1:TITLE$**

Use the **ð** cursor key to move the cursor to the 'i' of 'ion'. Then overtype the unwanted characters with spaces and the field will contain only the required words.

Equally, with the cursor on the 'i', tG pressed 3 times will remove the 3 offending letters.

When ENTER is pressed, the contents of the field will be written back to the record.

Tech note: The field variables from the current record are held in a temporary buffer while being manipulated. When the record pointer is moved, with a SKIP, GO, FIND, LOCATE or any other command which moves the pointer, that buffer is written back to the file. The file is not closed until a USE is issued or a QUIT. Therefore, if a fault happens when a file is open, some loss of data may occur.

Using command files

Within a command file, the GET command will be the more common method of changing variable values. If a variable contains non-blank data, that data will be displayed when the GET command is issued. That data may then be over-typed and replaced with the required new data. e.g.

```
LET confirm$="Y"
AT 4,13 SAY "Confirm this record to be added to database. Y/N"
AT 4,64 GET confirm$
```

At position 4,64 on the screen, the letter 'Y' will be displayed with the flashing cursor on it. If the ENTER key is hit, or the letter 'Y', the value of confirm$ will remain as 'Y'. If any other letter or number is hit, that will be the new value of confirm$.

Similarly for a field variable,

```
AT 4, 1 SAY "Enter type code"
AT 4,18 GET 1:TYPECODE$
```

The current value of TYPECODE$ in the current record will be displayed ready for overtyping. Any data left in the field and not overtyped, i.e. to the right of wherever the new data ends will have to be overtyped with spaces otherwise it remains in the field.

When used in conjunction with a 'G', the character under the cursor will be deleted. Note that the t key must be held down while pressing the 'G' in order for this to work. It is similar to the CONTROL function on a PC.

Other useful editing commands are tT (delete word), tD (delete to end of line) and t Y or t **-DEL-** (delete line).

A good example of an amending routine is SCSTOCK.PRG. It is part of the Stock Check system.

Deleting data

As with many database programs, deletion of data is done in a two part step. In the first stage, any records to be deleted are marked as such using the DELETE RECORD command. At a later time, when convenient, the file is copied across to a second file, leaving out the records marked for deletion.

The series of command lines would look like this.

- First, select the records for deletion and mark them with a delete flag.

```
}USE VIDEO
[Select record using LOCATE, FIND, GO etc]
}DELETE RECORD
}? DELETED()
1
[Select others and repeat]
```

When the deleted records are to be removed, the data file is renamed to a temporary name and then all the undeleted records are copied to a new file with the old name. viz.

- Close any open file

```
}USE
```

- Rename the file involved to a temporary name

```
}RENAME VIDEO TO TEMP
```

- Open the file with the temporary name

```
}USE TEMP
```

- Copy all records not marked for deletion

```
}COPY TO VIDEO FOR DELETED()=0
```

- Close the temp file

  }**USE**

- Delete the temp file

  }**DELETE FILE** TEMP

  The newly created file VIDEO will only contain those records which were not marked for deletion.

  See also the PACK.PRG file.

Selection and control

One of the prime functions of a database language is not only to facilitate easy input and amending of data, but also easy retrieval of selected bits of the data.

The following commands show how such retrieval works.

At curly prompt

```
FIND <"field value">
FIND <memory variable>
LOCATE FOR <condition>
CONTINUE
LIST [FOR <condition>] [FIELDS <fieldname,fieldname>]
DISPLAY [FOR <condition>] [FIELDS <fieldname,fieldname>]
```

When using an indexed file, the fastest way of locating a specific match on the key field is to use FIND. Although it has not been done on the Video file, it could have been indexed on the TYPE$ field.

For the sake of this demonstration, index the file on TYPE$.

```
}USE VIDEO
}INDEX ON TYPE$ TO TEMP.NDX
}FIND "COM"
}DISPLAY
10 005 The Woman in Red COM 005 102 5
}
```

Having found the first occurrence of this match, it is a relatively simple task to now list all the hits. Both the DISPLAY and LIST commands may be used.

```
}DISP FOR 1:TYPE$="COM"
10 005 The Woman in Red COM 005 102 5
11 006 Who's that girl COM 006 98 4
```

The curly prompt will not return immediately because the DISPLAY command is not intelligent enough to realise that an indexed file is in use. It continues searching the rest of the file for more hits. The --ESC- key stop the searching at any point.

LOCATE FOR .. CONTINUE

This command searches the file sequentially from the first record to the bottom of the file.

In the VIDEO system, the search might be for all the VIDEOS which are comedies i.e. with a TYPE$ of "COM". It would appear as

```
}LOCATE FOR 1:TYPE$="COM"
```

The next thing that will appear will be a new curly prompt with no message.
A **DISPLAY** command will display the current record. If an apparently blank record is displayed, that means that the end of the file has been found and there were no hits. Otherwise the first match will be displayed.

If the end of the file is found, the 'record number' enquiry will give a response of 0. viz

```
}? RECNO()
0
```

And the 'end of file' function will return a 1 for TRUE.

```
}? EOF()
1
```

If a hit has been found, the record number displayed will be the number of the hit. The EOF() function will return 0 for FALSE.

Having found a hit, the **CONTINUE** command will continue the search on the same basis. The search will continue from the current record to the bottom of the file.

LIST [FOR <condition>] [FIELDS <fieldname,fieldname>]
DISPLAY [FOR <condition>] [FIELDS <fieldname,fieldname>]

These two commands are very similar except that the **DISPLAY** command will only display records 7 at a time whereas the **LIST** command will scroll through all the hits without stopping.

To pause the scrolling data when using LIST, hold down the **-SHIFT- and** t keys at the same time.

On their own, each of these commands will display the current record.

The second part of the command line **[FOR <condition>]** will determine which records are displayed, i.e. only those which meet the condition will be shown.

}**LIST FOR 1:TYPE$="COM"**
will list all the records from the current record down the file for which the field 1:TYPE$ contains "COM".

Having said that, it may be that only the type, title and tape number are wanted. In this case the third part of the command is available which enables selection of the fields to be shown. e.g.

}**LIST FOR 1:TYPE$="COM" FIELDS TYPE$,TITLE$,TAPE_NO$**

In command files

The DISPLAY, LIST and ? may be used within a command file in order to provide structured output to the screen. The operator may be prompted to input the piece of data to be matched and the command file could then display that data in a structured fashion.

Similarly, the programmer may take complete control of how the data is displayed and which fields are displayed by using a command file as shown below.

The whole of the DO WHILE loop could be substituted with a single line, namely

DISPLAY FOR &fld$=fldval fields REF_NUM$, TITLE$, TYPE$, TAPE_NO$, DURATION$

Because the DISPLAY ALL command displays 7 lines at a time, it removes the need for the program to control the display lines and screen itself. Having said that, the flexibility of zBASE is demonstrated.

```
* zBASE Program to simulate display command
CLS
LET choice$="Y"
LET fld$=" "
LET fldval$=" "
LET onward$=" "
AT 0,10 SAY "Is the search to start at the top? Y/N."
AT 0,51 GET choice$
LET choice$=UPPER(choice$)
IF choice$="Y"
GO TOP
ENDIF
AT 1,10 say "Which field is to be searched ?"
AT 1,43 GET fld$
AT 2,10 say "What value for "+fld$+" is wanted."
AT 2,46 GET fldval$
CLS
AT 0, 2 SAY "Searching for " + fld$ + " = " + fldval$
LET line=1
```

```
LOCATE FOR &fld$=fldval$
DO WHILE EOF()=0
IF line>6
AT 7,1 say "Paused. Press any key to continue."
WAIT
cls
let line=0
endif

AT line, 2 say 1:REF_NUM$
AT line,10 say 1:TITLE$
AT line,32 say 1:TYPE$
AT line,37 say 1:TAPE_NO$
AT line,42 say 1:DURATION$
LET line=line+1
CONTINUE
ENDDO
AT line,1 say "End of file found."
RETURN
```

See also SCSIFIND.PRG.

Manipulating data files

Data is collected for a variety of reasons. Sometimes it's like an antique collection with bits of information collected so long ago that there is no relevance to that data. However, assuming the data is to be used, it must be available in different forms. This is especially true on the Z88 when it will often be used as a temporary home for data to be transferred to another micro.

Hence, the COPY and APPEND commands are available to produce files of different formats. These formats are zBASE, PipeDream columns, and comma delimited.

The zBASE format is a straightforward database file which can be directly accessed by zBASE. The COPY TO <filename> command would create an exact copy of the currently selected file. e.g.

```
}USE VIDEO
}COPY TO VIDBACK
}
```

PipeDream format, with the PD option at the end of the line, would create a file with each record on a line with TAB characters between fields. To load the file in PipeDream it must be loaded as Plain Text. Any fields of width greater than 12 will appear to be condensed. They are still there but the individual column widths will have to be reset to the required width in order to get all the data to appear.

To do this, load the newly created PD file under PipeDream. Enter Y to the Load as plain text prompt. Alter the width of each column in turn using ◇W. For the VIDEO file, the first column should be 3 characters wide, plus a couple for neatness. To set a width of say 5, position the cursor in column A and enter ◇W. Set new width to 5 and press ENTER This will set column A to 5 characters wide.

Press TAB and position the cursor in column B. ◇W then 22 then ENTER will set the title column width to 22. Each column can then be set to its required width for use under PipeDream.

Delimited means that the file is created for access by PipeDream with each record separated by a carriage return and each field is separated by a comma (,). This format is of particular interest to those wishing to export data to other database systems outside the Z88. Programs like dBASE II, III, IV and WordStar can use comma delimited files as raw data.

COPY TO filename [PD] [FOR <cond>][DELIMITED] [FOR <cond>]

COPY TO <filename> This version creates a straight copy of the file in zBASE format. e.g.

```
USE VIDEO
COPY TO VIDBACK
```

COPY TO <filename> PD

The whole file is copied to a PipeDream column format to be loaded as Plain text and the column widths adjusted as explained in the introduction to this section.

COPY TO <filename> DELIMITED

The whole file is copied but with commas indicating the start and end of each field and carriage returns indicating the end of each record. e.g.

```
USE VIDEO INDEX VID.NDX
COPY TO VIDCOM DELIMITED
```

The resulting VIDCOM file, when loaded as PLAIN TEXT under PipeDream, would look like this:

```
003,Bytes of Affection, BLU, 003, 0.45,C
004,Newest one on the bl, THR, 004, 1.00, B
005,Return of Zorro, OLD ,004, 1.50, F
002, Rubber dub dub, COM ,002 ,2. 75,B
001,Swedish Blu,BLU,001,3.5,A
```

In each of the cases described above, selected records may be copied by inserting the FOR phrase with a condition,

e.g. **FOR 1:TYPE$="COM"**

This becomes an extension to the retrieval routines available and constitutes another way of retrieving selected data while transferring it into another form.

Note that all uses of field variables must have the usage area prefix. If this prefix is left out, zBASE will only search for a memory variable to satisfy the condition. The only exception to this rule is when field names are given in a field list for DISPLAY FIELDS, or LIST FIELDS.

Report generation via PipeDream could be handled on a selective basis by using the COPY TO FOR line with the PD qualifier. e.g.

}**COPY TO VIDTHR PD FOR 1:TYPE$="THR"**

If the file to be copied to has been created previously, the COPY TO command will overwrite it WITHOUT WARNING.

APPEND FROM filename [PD] [DELIMITED] [FOR <cond>]

The file in use must be open in the PRIMARY database area and the SECONDARY area must be empty i.e. it must be CLOSED. (See DISPLAY STAT for how to check whether an area is in use.) The file from which data are to be appended must not be open.

This option is a reversal of the copy to command. To use APPEND, the file to receive the data must be open in the PRIMARY area. Data may then be obtained from another file and read into the current file, adding the new data to the bottom of the file.

The PD qualifier means that if a PipeDream columnated file has been saved as Plain text, each line will be treated as a database record and read into the file.

If the DELIMITED option is used, there can be no selection. The whole file will be appended at the bottom of the existing data.

The FOR <cond> option is only available when appending from another zBASE file. In this case each record in the source file is checked against the condition. If the condition evaluates to TRUE, or to a NON-ZERO value, the record will be added to the open target file.

COPY TO filename <[STRUCTURE] / [PD]>

This form of the copy command provides the opportunity to re-create the structure of the current file in another database file, or as a skeleton in PipeDream for the creation of a new file.

Such commands will be useful if a file structure has to be amended. Say the structure of the VIDEO file were to be amended in order to have an additional field called 'COST'.

}USE VIDEO
}DISPLAY STRUCTURE
REF_NUM STRING 3
TITLE STRING 20
TYPE$ STRING 3
TAPE_NO STRING 38
DURATION STRING 5
RATING STRING 1
}COPY TO TEMP STRUCTURE PD
}

n P and into PipeDream

Load TEMP as plain text. Go to the bottom of the file and add a line

COST

Save it as TEMP answering Y to the Plain Text question.

Then a nW will return to zBASE.

}**USE** (To close the current VIDEO file.)
}**RENAME VIDEO TO VIDEO.OLD**
}**CREATE VIDEO FROM TEMP**
}**APPEND FROM VIDEO.OLD**
}**USE**

The new file VIDEO now contains all the records from the previous VIDEO file with an additional field called COST of type NUMERIC.

COPY TO <filename> STRUCTURE

This version of the command will create a new database file with the same structure as the current file. The new file will have no records.

Moving around a file

These commands position the record pointer to the selected record. If the record number is known, the 'GO record number' command is the fastest method of reaching a record.

SKIP

The SKIP command moves the record pointer in the direction given in the command. SKIP 3 will move the pointer on three records. SKIP -3 will move the pointer back three records.

If the end of file is reached, the EOF() function will return a non-zero value, i.e. TRUE, and the record number, RECNO() will return a 0. SKIPping past the end of file will return RECNO()=0 and EOF()=1.

GO (TO RECORD NUMBER ..) <expression>

The GO number command places the record pointer at that record number. The number must be a numeric variable or numeric expression, except when it is BOTTOM or TOP.

GO TOP, GO BOTTOM

These commands position the pointer to the respective position at the top or bottom of the file. The record to which it is pointed is a live record and is not in front of the beginning of the file nor is it after the end. i.e. GO BOTTOM will go to the last record not to the end of file so EOF() will return FALSE.

ENVIRONMENTAL COMMANDS

-ESC- ON/OFF (v1.3 only)

The switch allows programs to ignore attempts to -ESC- and is activated by typing Set Esc=Off (deactivated by Set Esc=On) in interactive mode or by inserting it to a command line.

SET ECHO

This is a toggle command in that the first call will switch it on and the second will switch it off.

It is a programmers tool to show what command line is being executed. It is most use when debugging command files. It makes a mess of the screen but displays each command line as it is executed.

Indirect variables

&memvar

In dBASE II, these are referred to as MACROS. These & characters mean that the variable which follows the & is evaluated before being executed. In any command line from the keyboard or a command file, in which an indirect variable is found, the &variable is first replaced by the value it represents. e.g.

```
LET BETA=21
LET ALPHA$="BETA"
? ALPHA$
BETA
? &ALPHA$
21
```

If a command line was as follows, it would be evaluated in two steps viz:
GO &ALPHA$

would first be evaluated to
GO BETA

which would be executed as
GO 21

In any real program, the above would be written as GO BETA and the above is given as an example only. For a more extensive example of the use of indirect variables, see the APPEND.PRG program.

"&numericvar"

This is a particular use of the indirect variable and converts a numeric to a string.

```
LET numvar=42
? "&numvar"
42
? len("&numvar")
2
```

zBASE output to printer port

The command for this is
# "text"
or

# memvar$
or

# memvar$+" "+memvar2$ etc.

The line is sent to the printer exactly as organised on the command line.

See section C for a program for printing address labels.

Z88 output facilities

The copy commands explained above provide an opportunity to output to PipeDream. The method described below permits all the screen output to also go to a file or the printer.

n+ P

This sequence echoes all screen output to an attached printer. Keyboard input is double spaced at the printer.

n- P turns off this echo.

n+ S

In a similar fashion to the above printer echo system, this sequence echoes the screen output to a file. The file may be accessed via PipeDream by loading the file **ram.-/s.sgn**.

The switch to turn off the file echo is n - S.

System Limits

- The maximum number of records permitted in a zBASE file is 65535.
- The maximum command line length is 255.
- The maximum string variable length is 255. (Memory or field).
- Maximum number of fields per record is 32.
- Max number of nesting of IF/ENDIF loops is 255.
- Max number of nested DO WHILE's is 32
- Max number of nested DO files is 16
- Max area for memory variables is 512 bytes
- Max length of variable name is 8 characters
- Numbers are significant to 9 digits.
- Max number of decimal places is 8.

Precedence of Operators

zBASE does not consider one type of operator any more important than another. All mathematical expressions are evaluated from left to right. The only exception to this rule is that the contents of brackets are evaluated first. Programmers will quickly notice that expressions with brackets are evaluated relatively slowly, and that sorting out the expression will make the program run faster.

Using multiple databases.

There are many instances in data manipulation when data from two or more databases has to be combined to achieve the desired output. zBASE supports this requirement by allowing two databases, with indices, to be open at the same time. They can then cross reference each other and find the required information without opening and closing the files.

Continuing the home video library system, we decide to hold the names of the stars of each film, so if a film starring, say, Clint Eastwood, is wanted, a list of all Clint Eastwood films can be found quickly.

The most obvious way to do this is to add another field to the database: STAR$,20. However, a lot of films have more than one star. Many have three or four big names, so our database must have at least four STAR fields. This is going to be a waste of storage space for the films which only have one star, so another solution is needed. The best way to do it is to have a separate file holding the stars names.

This separate file will be linked to the VIDEO file via a reference number. Each video has a unique number, created when the video is first entered into the database. This number will be stored in each related record in the VIDSTAR file.

The format of the VIDSTAR file is very simple:

```
REF_NUM$,3
STAR$,20
```

Create the file and its index from zBASE using the file VIDSTAR.DEF as above and the following commands:

```
CREATE VIDSTAR FROM VIDSTAR.DEF
INDEX ON REF_NUM$ TO VIDSTAR.NDX
```

There will be one record in the VIDSTAR file for each star in the film. So if the film has three stars, the VIDSTAR file will have three records, where the REF_NUM is the same as the REF_NUM of the films record in the VIDEO file, and the STAR field holds the name of one of the three stars. An example will help:

VIDEO FILE RECORD		STARS FILE RECORDS	
REF_NUM$	12	REF_NUM$	STAR$
TITLE$	Trading Places	12	Eddie Murphy
TYPE$	COM	12	Dan Aykroyd
TAPE_NO$	3	12	Jamie Lee Curtis
DURATION$	112		
RATING$	5		

The data for the VIDSTAR file must be entered when the other details of the video are being entered. To alter the program VIDINP.PRG to do this, find the line in the program:

```
* FLAG A
```

and replace it with:

```
DO STARINP.PRG
```

The simple subroutine below, STARINP.PRG, should then be entered using PipeDream and saved as plain text.

```
* STARINP.PRG V1.00 By Derek Fountain
CLS
USE VIDSTAR INDEX VIDSTAR.NDX
AT 0,28 SAY "Home video library system"
LET header$="Enter the stars who appear in "+TRIM(mtitle$)
AT 2,(80-LEN(header$))/2 SAY header$
AT 6,25 SAY "Leave the field blank to exit"
```

The screen is cleared and the stars database VIDSTAR, with its index is selected. The program title is printed, then the string header$ is created with the TRIM of the video title. This string is then printed centrally on screen.

```
DO WHILE 1=1
LET mstar$=" "
AT 4,30 GET mstar$
IF mstar$=" "
USE VIDEO
CLS
RELEASE mstar$,header$
RETURN
ENDIF
AT 4,25 SAY "Creating record - Please wait"
APPEND BLANK
LET 1:REF_NUM$=mref_num$
LET 1:STAR$=mstar$
AT 4,25 SAY " "
ENDDO
```

The rest of the program is contained within an infinite loop, which goes round and round accepting as many star names as the user wants to enter. The stars name is entered into the variable mstar$, and is checked for a blank entry. If it is blank, the VIDEO file is reselected, the local variables RELEASEd and control passed back to VIDINP.

If the entry is not blank a new record is created in VIDSTAR. That record is filled with the unique reference number of the video, as created by VIDINP.PRG, and the stars name, as input by the user.

Run the program (DO VIDINP.PRG) and enter the details of a few films, along with the names of any stars. As many stars as required may be entered, including none.

Now, by using the REF_NUM from the VIDEO file as a key, it is a simple matter to find the associated records, i.e. the stars, in the VIDSTAR file.

This can be done interactively:

```
}SELECT 2
}USE VIDSTAR INDEX VIDSTAR.NDX
}SELECT 1
}USE VIDEO
}LOCATE FOR 1:TITLE$="Trading Places"
}DISPLAY
3 12 Trading Places 3 112 5
}SELECT 2
}LIST FOR 2:REF_NUM$=1:REF_NUM$
5 12 Eddie Murphy
6 12 Dan Aykroyd
7 12 Jamie Lee Curtis
```

or a program can use the same technique. The report program VIDREP.PRG demonstrates this. This program creates a report file for use with PipeDream. The routine uses another database called REP. The REP database only has one field:

LINE$,100

Each record of this database will contain one line of the report, which will be loading into, and printed from, PipeDream. This means three databases have to be manipulated:

VIDEO	The main database containing details of the video.
VIDSTAR	+ VIDSTAR.NDX. Holds details of the stars, indexed on reference number so they can be found quickly.
REP	Contains the lines of the report and is built up by the program.

The REP file is open from start to finish in area 1, and the other two are opened and closed as necessary in area 2.

```
* VIDREP.PRG V1.00 By Derek Fountain
CLS
SELECT 1
CREATE REP FROM REP.DEF
APPEND BLANK
LET 1:LINE$="TAPE TITLE/STARRING TYPE DURATION RATING"
```

The REP database is created in area 1. Its first record is a title line.

```
SELECT 2
USE VIDEO
```

Initially, the VIDEO database is opened in area 2.

```
AT 0,28 SAY "Home video library system"
AT 2,20 SAY "Please enter the name for the report file"
AT 6,25 SAY "Leave the name blank to abandon"
LET filename$=" "
AT 4,35 GET filename$
```

Get the required output file name into the mvar filename$.

```
IF filename$=" "
USE
SELECT 1
USE
DELETE FILE REP
RELEASE filename$
RETURN
ENDIF
```

If the filename was left blank, tidy up and exit.

```
CLS
AT 0,28 SAY "Home video library system"
AT 3,25 SAY "Creating workfile - Please wait"
```

Print a friendly message saying what's going on.

```
DO WHILE EOF()=0
This means DO WHILE the VIDEOs database in area 2 is not at EOF.
SELECT 1
APPEND BLANK
SELECT 2
LET 1:LINE$="
"+2:TAPE_NO$+" "+2:TITLE$+" "+2:TYPE$+" "+2:DURATION$+"  "+2:RATING$
```

Make a new record in area 1 (the REP database) and fill it with the basic information on the video. The VIDEO database is reselected.

```
LET key$=2:REF_NUM$
LET loc=RECNO()
```

Note the unique reference number of this video, and its record number in the file.

```
USE VIDSTAR INDEX VIDSTAR.NDX
FIND key$
```

Still in area 2, close the VIDEO database and open up the VIDSTAR database with its index. Locate the first occurrence of the reference number.

```
DO WHILE 2:REF_NUM$=key$
SELECT 1
APPEND BLANK
SELECT 2
LET 1:LINE$=" "+2:STAR$
SKIP
ENDDO
```

Because of the index, all the reference numbers will be grouped together, i.e. 12,12,12,13,13,15,16,16 etc. So go round a loop while the records key field matches the key field of the video. If the video has no entries in the stars database, the FIND will have left the pointer at EOF and this loop will not be entered.

The loop selects the REP database, adds a record to it, reselects the stars database, and fills the report line with the stars name. The next star is skipped to, and if it is the same video, the process starts again.

SELECT 1
APPEND BLANK

Reselect the REP file and add a blank line between this and the next video.

SELECT 2
USE VIDEO
GO loc

Reopen the VIDEO database in area 2 and return to the record which has just been dealt with.

SKIP
ENDDO

Move on to the next record, and repeat the process for each video on file. When the end of the file has been reached, move on.

AT 3,23 SAY "Creating report file - Please wait"
USE
SELECT 1
GO TOP

Close the VIDEO database in area 2, select area 1 and move to the top of the REP database.

COPY TO &filename$ PD
USE
AT 3,23 SAY "Report file created - Press any key"
WAIT
DELETE FILE REP
RETURN

Copy out the file to a PipeDream file, close the database, and wait for acknowledgement that the process has finished. Then delete the work file and return.

Importing files from PipeDream

When data has been entered previously under PipeDream, it can be imported directly into zBASE using the APPEND FROM command. Assuming the PipeDream data are arranged in columns, separated by **TAB** characters, each line of data can be read in as a record. Each column is then a field.

The phone book from the Quick Start Guide might look like this in PipeDream. It is called PH.DAT and is stored as PLAIN TEXT.

	................A	..............B	C
1	Cambridge	312216	Sir Clive
2	Rakewell	630617	Vic Gerhardi
3	UserClub	68 Well St	Roy
4	Scotland Yd	877 1212	Insp Bond

The section has a structure of

COMP$,15; PHONE$,17; NAME$,20; KEYFIELD$,5

The keyfield is explained in the QSG. For these purposes it may be ignored.

With the above file definition in a PipeDream file, the required database file may be created in zBASE. Once created, the above data may be imported direct from PipeDream into the new zBASE file.

The command line is:

}**APPEND FROM PH.DAT PD**

All the data from the above file will then be read into the zBASE file setup above.

Fields which are too big for the database field sizes specified will be trimmed to size by lopping off the last characters.

Editing command files

Editing command files and instantly testing them is extremely simple. It can be done by using **□P** to go into PipeDream, and **□W** to return to zBASE.

Popdowns from zBASE.

Z88 popdowns may be used while in zBASE. However, the following points should be noted.

If a popdown is called when editing a command line, that command line will return a SYNTAX ERROR when zBASE is re-entered.

If the **-INDEX-** key is used to leave a popdown, zBASE will appear as a suspended application. It should not be KILLed.

Editing command files and instantly testing them is extremely simple. It can be done by using nP to go into Pipedream, and nW to return to zBASE.

zBASE Commands

Most of the zBASE commands may be used both in the interactive mode at the curly prompt, or in program files or command files. The main exceptions to this rule are that related pairs of commands such as **IF/ENDIF** and **DO WHILE/ENDDO** may not be used in the interactive mode.

This section of the manual lists each command and its syntax and defines those commands which are restricted.

The symbols <...> bracket items that are to be specified by the user. Square brackets [...] enclose optional items.

exp... An expression which can result in either a number or a string. e.g.

5+5,
"FRED "+"BLOGGS",
a$+"MUMMY".

var ... A variable, can mean either mvar or fvar.

mvar ... A memory variable, not stored in a database, but in a large buffer in RAM. A mvar is defined as a string if its name ends with a '$', otherwise, it is defined as a number.

fvar ... A field variable, permanently stored in a database. Fields must start with either 1: or 2: label, depending on the database they are to be taken from. If the label is missing the field will be taken as a mvar.

cond ... A condition which returns the result either TRUE or FALSE. e.g.

10=10 is TRUE,
10=6 is FALSE,
"FRED"="BLOGGS" is FALSE.

A single number 0 is evaluated as FALSE, any other number is TRUE.

At present only a single condition is allowed. i.e. there are no AND, OR or NOT clauses.

KEYWORDS

Only the first 4 characters of a keyword are significant.

* Comments

Any line beginning with an * will be ignored. This can be used as a method of putting comments in command files. e.g.

```
}* This is a comment line and will be ignored
}
```

### ? [<exp>]

This command shows the value of an expression. The result is printed at the current cursor position, with no CR or LF following.

If [<exp>] is not supplied a CR LF sequence is sent to the screen. e.g.

```
}? "Hello world"
Hello world
}? 10+10
20
}? 1:phone$
01296 43 78 78
```

# [<exp>]

This command works just like the ? command, only the expression is sent out of the serial port, instead of to the screen.

AT <co-ordinates> SAY <exp>

This command prints the value of the expression at the specified co-ordinates. The co-ordinates are taken as 'down' (0-7) then 'across' (0-79). A string will be printed in full, including trailing spaces and a number will be printed to 9 significant digits and up to 8 decimal places. e.g.

```
}AT 3,40 SAY "Merry Christmas"

Merry Christmas
```

AT <co-ordinates> GET <var>

This command allows the input of a variable, either field or memory variable. The co-ordinates are taken as for AT SAY with the variable then being offered up at the specified co-ordinates for editing. The variable must exist prior to the issuing of the GET command.

If the variable is a string the number of characters in the string will determine the number of characters to be edited. e.g. if a$ is "<10 spaces>", then GETting a$ will allow editing of 10 characters.

If the variable is a number then 9 characters can be edited, with any one being a decimal point. e.g. if a is 0, the GETting a will allow editing of 9 characters, all of which must be numerics, except one which can be a decimal point in any position. This way the user can decide the number of decimal places. If anything other than a numeric is entered the number will not be accepted. The GET will remain active and the number will have to be re-entered.

```
LET mvar$="0101 212 800 5555"
AT 3,30 GET mvar$
```

APPEND [BLANK] / [FROM <filename> [PD]/[DELIMITED] [FOR <exp>]]>

This command is used to add new records to the currently selected database.

If the BLANK clause is specified a single blank record is added to the end of the currently selected database. All the fields will be empty and the record pointer will be left pointing to the new record. e.g.

```
}GO BOTTOM
}? RECNO()
24
}APPEND BLANK
}? RECNO()
25
```

The FROM clause causes records in the file 'filename' to be added to the bottom of the currently selected file. e.g.

```
}USE PHBOOK
}APPEND FROM PHBOOK.NEW
```

will add all the records in PHBOOK.NEW to the bottom of the PHBOOK file.

The PD clause means the fields are separated by TAB (09h) characters and the records are separated by CR (0Dh) characters.

The command line is

APPEND FROM [filename] PD

The file shown below, having been saved as PLAIN TEXT, will then be appended onto the end of the database.

	................A	..............B	C
1	Cambridge	312216	Sir Clive
2	Rakewell	630617	Vic Gerhardi
3	UserClub	68 Well St	Roy
4	Scotland Yd	877 1212	Insp Bond

The DELIMITED clause means the fields are separated by commas and the records by CR (0Dh) LF (0Ah). This is a common format output by dBase and many spreadsheets.

The command line would look like:

APPEND FROM [filename] DELIMITED

The following file would be appended to the database in use.

	................A	..............B	C
1	Cambridge,312216,Sir Clive		
2	Rakewell,630617,Vic Gerhardi		
3	UserClub,68 Well St,Roy		
4	Scotland Yd,877 1212,Insp Bond		

If the PD and DELIMITED instructions are omitted the file is assumed to be a zBASE file.

CLS

This command clears the screen.

CONTINUE

This command locates the next record which matches the condition specified in the last LOCATE command.

The search resumes at the current record, and continues until a match is found or EOF. e.g.

```
} LOCATE FOR 1:NAME$="Fred"
} ? RECNO()
14
} DISPLAY
Fred Bloggs
} CONTINUE
} ? RECNO()
24
} DISPLAY
Fred Smith
```

The CONTINUE command is not designed for use with the FIND command. If CONTINUE is used after a FIND it is only coincidence if matching records are found.

COPY TO <filename> [PD] [DELIMITED] [FOR <cond>]

The copy command is the method of outputting data to another format. If none of the options are specified the output file will be another zBASE file.

If PD is specified the data will be written out in a format which PipeDream can read in plain text mode.

e.g. **COPY TO <filename> PD**

If DELIMITED is specified the data will be written out in a comma delimited format which can be read in by dBase or a Spreadsheet package. e.g.

COPY TO <filename> DELIMITED

If the FOR clause is specified only those records which meet <cond> will be copied to the output file. e.g.

COPY TO <fileother> FOR 1:PHONE$="01"

will only copy records with a phone$ field starting with "01".

COPY TO <fileother> PD FOR 1:PHONE$="01"

copies the same set of records but this time the output is in PipeDream format. Similarly, the use of DELIMITED would out to a comma delimited file.

COPY TO <filename> [STRUCTURE] [DELIMITED]

This second version of the COPY command provides two alternative outputs.

COPY TO <filename> STRUCTURE, creates a zBASE file with no records but with the same structure as the current file.

COPY TO <filename> STRUCTURE DELI, creates a PipeDream file of the form required to create a new zBASE file. The new file created from such a file would have the same structure as the original database from which the structure was taken.

CREATE <file1> FROM <file2>

This version of the CREATE command creates a database file called 'file1' from an input file, 'file2' which is created under PipeDream. The format for file2 must follow these rules:

1. One line per field in database.
2. No blank lines, and no extra CR at the end of the last field.
3. All string field names end in $, followed by a comma and a number indicating the length of the field (1-255).
4. Numeric fields have just the name on the line, nothing else.

e.g.

NAME$,25	NAME is a string field, length 25
AGE	AGE is a numeric field
SCORE$,10<END>	SCORE is a string field, length 10

Any stray CRs, or text in the wrong place may cause strange results.

The database file created will be left open in the selected database area. If the CREATE fails for any reason, the selected area is left empty.

DELETE <[RECORD] / [FILE <filename>] >

The DELETE RECORD command marks the current record for deletion. The record is not actually removed from the file. When the record is “DISPLAYed” a '*' will appear next to the first field indicating the record is marked for deletion.

To actually remove deleted records from the database, see the program PACK.PRG.

DELETE FILE filename will erase the file from the Z88's Ramdisk. If it doesn't exist, an error message will appear.

DISPLAY/LIST <[STRUCTURE]/[MEMORY]/[STATUS]>/ <[ALL]/[FOR <cond>]> <[FIELDS field list]>

The display commands can be called using the LIST keyword. The only difference in their operation is that DISPLAY pauses for each screenfull of data, and LIST doesn't.

If the STRUCTURE clause is used the structure of the file open in the selected database area is displayed on screen. It takes the format:

NAME$ STRING
AGE NUMERIC

If the MEMORY clause if used, the memory variables and their values are listed on screen. e.g.

address1$	STRING	"54 Moor Road"
address2$	STRING	"Linstanton"

If the STATUS clause is used the machines current status is displayed on screen.

The information provided by DISPLAY STATUS shows the name of the file(s) currently open pus information about record sizes and the current record number. It also shows which user area is active selected.

If none of the above options are specified then the current data is displayed.

DISPLAY on its own will print the contents of the current record to screen. e.g.

```
}DISPLAY
Fred Bloggs 49.0000 Painter
```

DISPLAY ALL will print the contents of all records on file, seven records at a time, starting at the current record. Press -ESC- to stop. e.g.

```
}DISPLAY ALL
Fred Bloggs 49.0000 Painter
Joe Clump 32.0000Electrician...
```

DISPLAY FOR <cond> will print the contents of all records which meet the condition e.g.

```
}DISPLAY FOR 1:FORENAME$="Fred"
Fred Bloggs 49.0000 Painter
```

A FIELDS clause can be added to either the DISPLAY ALL or DISPLAY FOR command so only the required data is printed. e.g.

```
} DISPLAY ALL FIELDS FORENAME$
Fred
Joe
Andrew
```

Note that the 1: is not required for the list of field names.

DO <command file>

This command executes the specified command file. All commands in the file are in standard zBASE syntax. Control is passed back to the keyboard when the last instruction has been executed, or a RETURN is encountered.

DO WHILE <cond> - ENDDO

The **DO WHILE** command can only be executed from within a command file. The condition is evaluated, and, if it is TRUE the statements following the DO are executed until an ENDDO is found. At that point control is passed back up to the DO and the condition is re-evaluated.

When the condition is FALSE control passes to the command after the ENDDO. Example:

```
LET x=1
DO WHILE x<>10
? "Hello world"
LET x=x+1
ENDDO
? "Goodbye"
```

will print out 'Hello world' 10 times followed by 'Goodbye'. Note the indentation of the commands within the loop. This makes the program much easier to understand, and while not mandatory, is very necessary in nested DO WHILE loops.

Also note the expression 'DO WHILE 1' would put the machine into an infinite loop.

FIND <exp>

Find searches the current index file for the value <exp>. It is zBASE's quickest method of finding data, searching some 4000 bytes per second. The only field it scans is the index key field. e.g.

```
}USE TEST
}INDEX ON NAME$ TO TESTNAME.NDX
}FIND "Derek Fountain"
}DISPLAY
465 Derek Fountain Aylesbury
}FIND "Aylesbury"
}DISPLAY
0
}INDEX ON TOWN$ TO TEST.NDX
}FIND "Aylesbury"
}DISPLAY
465 Derek Fountain Aylesbury
```

GO [<exp>/<BOTTOM>/<TOP>]

The GO command positions the current record pointer at an absolute record number, specified by exp. Trying to move to a number less than 1 will place the record pointer at 1, and trying to move to a record number which doesn't exist will place the record pointer at the last record in the file (not EOF).

If the BOTTOM clause is specified the record pointer will move to the last record in the file.

If the TOP clause is specified the record pointer will move to the first record in the file. e.g.

```
} ? RECNO()
24
} GO 10
} ? RECNO()
10
} LET a=30
} GO a
} ? RECNO()
30
} GO BOTTOM
} ? RECNO()
98
} GO TOP
} ? RECNO()
1
```

IF <cond> - ELSE - ENDIF

The IF command allows conditional execution of statements from within a command file, but without the looping system the DO WHILE statement uses.

The condition is evaluated, and if it is TRUE, the statements following the IF are executed. If the condition is FALSE, control passes to the statement following the next ENDIF, or the ELSE if there is one. E.g.

```
IF a$=b$
? "Hello world"
ENDIF
? "Goodbye"
```

Also:

```
IF a$=b$
? "It’s equal"
ELSE
? "It’s not equal"
ENDIF
? "Goodbye"
```

In the first example, if a$ is the same as b$ then 'Hello world' will be printed, followed by 'Goodbye'. If a$ is not the same as b$, then just 'Goodbye' will be printed.

In the second example, if a$ is the same as b$ then 'Its equal' will be printed, followed by 'Goodbye'. If a$ is not the same as b$, then 'It’s not equal' will be printed, followed by 'Goodbye'.

INDEX ON <fvar> TO <filename>

Index is zBASEs method of sorting a database, and keeping it sorted. When a database is indexed on a field, any records that are appended to it automatically take up their correct place in the file. If the file is USEd without the index file, the records go back to chronological order. E.g.

```
}USE TEST
}LIST ALL
1 Fred
2 Bob
3 Peter
}INDEX ON NAME$ TO TEST.NDX
}GO TOP
}LIST ALL
1 Bob
2 Fred
3 Peter
}APPEND BLANK
}LET 1:NAME$="Derek"
}GO TOP
}LIST ALL
1 Bob
2 Derek
3 Fred
4 Peter
}USE TEST
}LIST ALL
1 Fred
2 Bob
3 Peter
4 Derek
```

LET <var>=<exp>

The LET command assigns a value to a specified variable. In the generic case, the value to be allocated to the variable is the value of the expression <exp>. The label to be given to the variable is the name contained in the space <var>. e.g.

```
LET today$=date()
LET spaces$=" "
LET codename$="Fred"
```

LOCATE FOR <cond>

The locate command is used for finding a record which meets a specified condition. The record pointer will be moved to the top of the file and each record checked to see if <cond> is true. If it isn't the next record is checked and so on until the EOF is found or a record meets the condition. The record pointer will be left pointing to the correct record, or EOF if there wasn't one. e.g.

```
} ? RECNO()
24
} LOCATE FOR 1:NAME$="Fred"
} ? RECNO()
14
} LOCATE FOR 1:NAME$="Gertrude"
} ? EOF()
1
```

QUIT

This command closes all open files, and returns the Z88 back to its main applications menu.

RECALL RECORD

This command performs the opposite of the delete command. A record marked for deletion will be 'unmarked'. If the record wasn't deleted, the command is ignored.

RELEASE <mvar>

When a memory variable has served its purpose and is no longer needed, it can be removed from RAM to free space. e.g.

```
} RELEASE name$
} ? name$
Variable not found
```

RENAME <file1> TO <file2>

This command renames file1 to file2 in the same way the option from the Filer does. If file2 exists or file1 doesn't an error message will appear. Note you can't rename a file which is open.

RETURN

Return halts the execution of the current command file and returns control back to the curly prompt or the calling file.

All files are left open and memory variables are maintained.

SELECT <[1 or 2]>

zBASE can have two databases open at the same time, for cross referencing purposes. This command selects between the two database areas and defines which database will be used for DISPLAY commands, and which area a new database will be opened in when a USE command is issued.

The two areas are called PRIMARY and SECONDARY or simply 1 and 2. Only the 1 or 2 may be used in the actual command.

SKIP [<exp>]

The skip command moves the current record pointer. If <exp> is specified it must be a number. The record pointer will move that number of records through the file, from its current position. A negative number will make it move backwards. Trying to move past the beginning of the file will leave the record pointer at record 1, and trying to move past the end of the file will set EOF true.

If no <exp> is given the record pointer will move one record forwards. e.g.

```
} ? RECNO()
24
} SKIP
} ? RECNO()
25
} SKIP 10
} ? RECNO()
35
} SKIP -4
} ? RECNO()
31
```

USE [<file>]

This command opens a new database in the currently selected area. If the filename is not specified the database currently open is closed and the area left empty. Only files created with the CREATE command can be opened using USE. i.e. only zBASE files.

The alternative form opens an associated INDEX file viz:

USE file INDEX file

WAIT

This command simply pauses the system until a key is pressed. When used in a command file it is useful to display a 'WAITING' message so that the operator knows that a key is awaited.

If this is done, the 'WAITING' message should be erased after a key press.

zBASE Functions

zBASE supports all the standard functions found in other languages plus some specialised database handling ones of its own.

Experienced programmers will notice there is no STR function to convert a number to a string. This is because the Indirect Variable system can be used to emulate this function. To emulate a STR function:

```
} LET A=2.03
} LET A$="&A"
} ? "-"+a$+"-"
-2.03-
```

CHR(<exp>)

The CHR function returns a single character string containing the character whose ASCII code is <exp>. e.g.

```
} ? CHR(65)
A
} ? CHR(124)
|
```

CLI (<exp$)

The CLI function sends <exp$> to the OZ CLI function for immediate execution.

DATE()

The date function returns the current date as an eight bit string. The date is taken from OZ, and is returned in the system default format i.e. European or American as set by the panel. e.g.

```
} ? DATE()

18/08/88
```

DELETED()

The DELETED function returns a 1 (TRUE) or a 0 (FALSE) depending on whether the current record is marked for deletion or not. e.g.

```
} DELETE RECORD
} ? DELETED() 1
} RECALL RECORD
} ? DELETED()
0
```

EOF()

The EOF (End of file) function returns a 1 (TRUE) or a 0 (FALSE) depending on whether the current file record pointer is at end of file or not. e.g.

```
} GO BOTTOM
} ? EOF()
0
} SKIP
} ? EOF()
1
```

FILE(<exp$>)

The FILE function returns a 1 (TRUE) or a 0 (FALSE) depending on whether the file <exp$> exists or not. Note that <exp$> is an expression and not a literal. e.g.

```
} ? FILE("NAMES.DBF")
1
} USE NAMES.DBF
```

INT(<exp>)

The INT (Integer) function removes all digits from the number <exp> which follow the decimal point. e.g.

```
} ? INT(3.142)
3
} ? INT(200.000)
200
} ? INT(22/7)
3
```

LEN(<exp$>)

The LEN function returns the length of string expression <exp>. e.g.

```
} ? LEN("HELLO WORLD")
11
} LET A$="FOO BAR ZOK POW"
} ? LEN(A$)
15
```

LOWER(<exp$>)

The LOWER function turns all the characters in the string <exp> into lower case. Any characters that were already in lower case will be left alone. e.g.

```
} ? LOWER("Fred Bloggs")
fred bloggs
```

LTRIM(<exp$>)

The LTRIM function removes all leading spaces from the string expression <exp>. e.g.

```
} ? "1"+LTRIM(" 2345")+"6"
123456
```

RAM()

reveals RAM space available on currently selected device.

RECNO()

The RECNO function returns the current file current record number. e.g.

```
} GO 10
} ? RECNO()
10
} GO BOTTOM
} ? RECNO()
98
} SKIP
} ? RECNO()
0
```

SET ECHO

toggles echoing of all commands to screen.

STR

To emulate a **STR** function use &.

```
} LET A=2.03
} LET A$="&A"
} ? "-"+a$+"-"
-2.03-
```

SUBSTR(<exp1$>,<exp2>,<exp3>)

The SUBSTR (Substring) function returns that part of <exp1$> that starts at <exp2> and is <exp3> characters long. Examples make this clearer:

```
} ? SUBSTR("1234567890",3,4)
3456
} ? SUBSTR("1234567890",8,2)
89
```

Note that if <exp2>+<exp3> > LEN(<exp1$>) no error occurs.

TIME()

The time function returns an eight character string holding the current time in the format HH:MM:SS. The time is fetched from OZ and can be altered using nT. e.g.

```
}? TIME()
16:28:37
```

To obtain time differences, sub strings of the time string will have to be extracted and converted to numbers using the VAL() function.

TRIM(<exp$>)

The TRIM function removes the trailing spaces from the end of string <exp>. e.g.

```
} ? "1"+TRIM("2345 ")+"6"
123456
```

UPPER(<exp$>)

The UPPER function turns all the characters in the string <exp> into upper case. Any characters that were already in upper case will be left alone. e.g.

```
} ? UPPER("Fred Bloggs")
FRED BLOGGS
```

VAL(<exp$>)

The VAL function turns an ASCII string into its numeric equivalent. Conversion stops at the first non ASCII digit character. e.g.

```
} ? VAL("123")
123
} ? VAL("10")+10
20
```

WHERE(<exp1$>,<exp2$>)

The where function returns the position of <exp1$> in <exp2$>. If it is not present, 0 is returned. e.g.

```
} ? WHERE("23","1234")
2
} ? WHERE("HELLO",UPPER("hello world"))
1
} ? WHERE("78","1234")
0
```

zBASE Programs

The programs listed in this manual are designed to be indications only of the potential of zBASE. They are for users to amend to suit their own purposes and are not intended as complete solutions. They have not been exhaustively tested.

Owners of a license to use zBASE may make free use of all the programs listed herein under the one condition that original authorship is acknowledged.

Those who wish to type in the programs by hand, be warned that some code lines have spaces in them which are not normally seen in these listings. The space character has been replaced by   unless the spaces are in a comment line. Take care that you count the number of spaces correctly.

These programs are in programs.doc which you can now download from this site.

Alternatively you can obtain these programs from Rakewell for the price of a blank Z88 EPROM. If preferred, a blank EPROM and £10 plus VAT handling fee may be sent to Rakewell for the programs to be put onto the EPROM and returned.

They may also be obtained on a PC disk for £5, plus VAT, direct from Rakewell.

Sample programs

These files may be copied and adapted by owners of a zBASE licence at no charge.

Wordmongers retains all copyright in them.

MAIN.PRG	A menu program for database management	
NEWUN.PRG	Called by MAIN to enter new records.	
APPEND.PRG	A command file to make data entry easier.	
APPEND.DEF	A database structure file for use by the APPEND program.	
PACK.PRG	To remove deleted records from a file	
PHENT.PRG	Phone book entry program	
FPROG	Find program for phone book	
VIDINP.PRG	Input routine for VIDEO file	
STARINP.PRG	Input for stars in Videos	
VIDREP.PRG	Report generator for video system	
VIDREP.DEF	USE REP.DEF FROM STOCK CONTROL SYSTEM	
Stock Check suite.	See introduction page for this suite.	

MAIN.PRG

```
* MAIN.PRG - A menu program for database file * management.
* TITLE.PRG
do while 1=1
 cls
 at 2,10 say "The Wordmongers zBASE Address book. By C Salvidge."
 at 3,15 say "(c) Wordmongers Ltd 1988."
 at 5,10 say "(S)earch for an entry. (E)nter new person. (Q)uit."
 at 7,10 say "Your choice please"
 let choice$=" "
 do while where(choice$,"SEQ")=0
  at 7,30 get choice$
  let choice$=upper(choice$)
 enddo
 if choice$="Q"
  return
 endif
 if choice$="S"
  do findum.prg
 endif
 if choice$="E"
  do newun.prg
 endif
enddo
```

NEWUN.PRG

```
* NEWUN.PRG - Called by MAIN to enter new records.
cls
at1,0say"Title:     Forename:        Surname:"
at 3,0say "Enter the name of the person to add " at 4,0 say "Leave all the fields blank to
cancel"
let title$="   "
let forname$="                 "
let surname$="                    "
at 1,5 get title$
at 1,20 get forname$
at 1,52 get surname$
if len(surname$)=0
 if len(forname$)=0
  if len(title$)=0
    cls
    ? "Cancelled Hit any key"
    wait
    return
  endif
 endif
endif
let usurnam$=upper(surname$)
let ufornam$=upper(forname$)
let looking$="Y"
locate for upper(1:surname$)=usurnam$
if eof()=1
  let looking$="N"
endif
go top
do while looking$="Y"
  continue
  if eof() <> 1
    if upper(1:forname$) = ufornam$
      cls
      ? "That person is already on the database-Try again"
      ? "Press any key to cancel"
      wait
      return
    endif
    if upper(1:forname$) <> ufornam$
     looking$="N"
    endif
  endif
  if eof() = 1
    let looking$="N"
  endif
enddo
append blank
let 1:surname$=surname$
let 1:forname$=forname$
let 1:title$=title$
at 2,0 say "Address                              "
at 4,0 say "                              "
at 3,0 say "                              "
at 5,0 say "Note"
at 2,10 get 1:add1$
at 3,10 get 1:add2$
at 4,10 get 1:add3$
at 5,10 get 1:note$
return
```

APPEND.PRG

```
* APPEND.PRG - A command file to make data entry easier.
* **********************************
* APPEND.PRG Will append new records *
* to the database that is open in 1: *
* **********************************
*
* The database to be appended to is open in 1
* First copy its structure out to a PD file
select 1
copy to temp.def structure PD
select 2
* Now, in 2, Create a database for the structure to be read in to
create st.dbf from append.def
* Read in the structure
append from temp.def PD
use st.dbf
let doing=1
do while doing=1
 select 1
 append blank
 cls
 at 0,0 say "Record number - "
 at 0,16 say recno()
 select 2
 go top
 do while eof()=0
  select 2
  at 7,0 say 2:field$
  let f$="1:"+2:field$
  at 7,10 get &f$
  skip
  ?
 enddo
 let choice$=" "
 do while where(choice$,"YN")=0
   at 7,0 say "Add another record ? Y/N"
   at 7,25 get choice$
   let choice$=upper(choice$)
 enddo
 if choice$="N"
   let doing=0
 endif
enddo
* Now clean up
select 2
use
delete file st.dbf
select 1
* Thats it folks
```

APPEND.DEF

```
* APPEND.DEF - A database structure file for use by the
* APPEND program.
* Do not enter the * lines to the append.def file in Pipedream.
field$,10
type$,3
```

PACK.PRG

```
* PACK.PRG Removes deleted records from file.
CLS
IF FILE(TEMP)
  AT 2,15 SAY "File called TEMP currently exists."
  AT 3,15 SAY "Please remove it.  PACK needs that "
  AT 4,15 SAY "file name as a temporary work file."
  AT 7,15 SAY "Press any key to return to }."
  WAIT
  RETURN
ENDIF
USE
LET SPACE40$="                                        "
LET NEWFILE$=SUBSTR(SPACE40$,1,12)
AT 2,15 SAY "Enter name of file to be packed.."
AT 2,50 GET NEWFILE$
AT 4,15 SAY "Renaming "+NEWFILE$+" to TEMP."
RENAME &NEWFILE$ TO TEMP
USE TEMP
COPY TO &NEWFILE$ FOR DELETED()=0
USE
DELETE FILE TEMP
RELEASE NEWFILE$, SPACE40$
RETURN
* EOF
```

PHENT.PRG

```
* PHENT.PRG
LET doing=1
* following line sets up variable of 20 spaces
LET SPACE20$="                    "
DO WHILE doing=1
```

```
LET COMP$=SUBSTR(SPACE20$,1,15)
LET PHONE$=SUBSTR(SPACE20$,1,17)
LET NAME$=SPACE20$
LET KEYFIELD$=SUBSTR(SPACE20$,1,5)
CLS
AT 0,10 SAY "ENTER Company name..."
AT 1,10 SAY "ENTER contact name..."
AT 2,10 SAY "ENTER phone number..."
AT 3,10 SAY "ENTER KEYFIELD VALUE."
AT 0,33 GET COMP$
AT 1,33 GET NAME$
AT 2,33 GET PHONE$
IF COMP$=" "
  LET KEYFIELD$=SUBSTR(NAME$,1,5)
ELSE
  LET KEYFIELD$=SUBSTR(COMP$,1,5)
ENDIF
AT 3,33 GET KEYFIELD$
LET confirm$="N"
AT 5,10 SAY "Confirm this record to be added to file. Y/N."
AT 5,57 GET confirm$
IF confirm$="Y"
  APPEND BLANK
  LET 1:COMP$=COMP$
  LET 1:NAME$=NAME$
  LET 1:PHONE$=PHONE$
  LET 1:KEYFIELD$=KEYFIELD$
ELSE
  AT 5,57 SAY "NOT APPENDED"
ENDIF
LET choice$=" "
DO WHILE WHERE(choice$,"YN")=0
  AT 7,10 SAY "Add another record? Y/N"
  AT 7,35 GET choice$
  LET choice$=UPPER(choice$)
  ENDDO
  IF choice$="N"
    LET doing=0
  ENDIF
ENDDO
RELEASE doing, SPACE20$, COMP$, NAME$, PHONE$,KEYFIELD$
RELEASE choice$, confirm$
RETURN
* END OF COMMAND FILE
```

FPROG

```
* FPROG - Programmed FIND routine for phone book
let SPACE20$="                    "
LET mseek$=SUBSTR(SPACE20$,1,5)
LET looping=1
DO WHILE looping
  cls
  AT 1,10 SAY "Find what??"
  AT 1,38 GET mseek$
  LET mseek$=TRIM(mseek$)
  IF LEN(mseek$)=0
    LET mseek$=" "
  ENDIF
  AT 2, 0 SAY " "
  IF mseek$=" "
    LET looping=0
  ELSE
    FIND mseek$
    IF EOF()=1
      AT 2,10 SAY "No find"
    ELSE
      DISP
    ENDIF
    AT 7,10 SAY "WAITING"
    WAIT
    AT 7,10 SAY "       "
  ENDIF
ENDDO
DISP
RELEASE SPACE20$, MSEEK$
RETURN
* EOF
```

VIDINP.PRG

```
* VIDINP.PRG V1.00 By Derek Fountain
CLS
USE VIDEO
GO BOTTOM
LET nref_num=VAL(1:REF_NUM$)
AT 0,28 SAY "Home video library system"
AT 2,51 SAY "Tape  :"
AT 4,16 SAY "Title :"
AT 4,51 SAY "Type  :"
AT 5,16 SAY "Length:"
AT 5,51 SAY "Rating:"
DO WHILE 1=1
  LET nref_num=nref_num+1
  LET mref_num$="&nref_num"
  LET mtape_no$="   "
  LET mtitle$="                    "
  LET mtype$="   "
  LET mduration$="   "
  LET mrating$=" "
  AT 2,16 SAY "Ref number: "+mref_num$
  AT 2,58 SAY mtape_no$
  AT 4,23 SAY mtitle$
  AT 4,58 SAY mtype$
  AT 5,23 SAY mduration$
```

```
 AT 5,58 SAY mrating$
 AT 7,23 SAY "Enter the information on the video"
 AT 2,58 GET mtape_no$
 IF mtape_no$="   "
   RELEASE nref_num,mref_num$,mtape_no$,mtitle$,mtype$,mduration$
  RELEASE mrating$,confirm$
   RETURN
 ENDIF
 AT 4,23 GET mtitle$
 AT 4,58 GET mtype$
 AT 5,23 GET mduration$
 AT 5,58 GET mrating$
 LET confirm$=" "
 DO WHILE confirm$=" "
   AT 7,23 SAY "Please confirm this information: "
   AT 7,56 GET confirm$
   IF WHERE(confirm$,"YNyn")=0
     LET confirm$=" "
   ENDIF
 ENDDO
 IF UPPER(confirm$)="Y"
   APPEND BLANK
   LET 1:REF_NUM$=mref_num$
   LET 1:TAPE_NO$=mtape_no$
   LET 1:TITLE$=mtitle$
   LET 1:TYPE$=mtype$
   LET 1:DURATION$=mduration$
   LET 1:RATING$=mrating$
   * FLAG A
 ELSE
   LET nref_num=nref_num-1
 ENDIF
ENDDO
```

STARINP.PRG

```
* STARINP.PRG V1.00 By Derek Fountain
CLS
USE VIDSTAR INDEX VIDSTAR.NDX
AT 0,28 SAY "Home video library system"
LET header$="Enter the stars who appear in "+TRIM(mtitle$)
AT 2,(80-LEN(header$))/2 SAY header$
AT 6,25 SAY "Leave the field blank to exit"
DO WHILE 1=1
  LET mstar$="                    "
  AT 4,30 GET mstar$
  IF mstar$="                    "    USE VIDEO
    CLS
    RELEASE mstar$,header$
    RETURN
  ENDIF
  AT 4,25 SAY "Creating record - Please wait"
  APPEND BLANK
  LET 1:REF_NUM$=mref_num$
  LET 1:STAR$=mstar$
  AT 4,25 SAY "                              "
ENDDO
* EOF
```

VIDREP.DEF

```
* VIDREP.DEF
LINE$,100
```

VIDREP.PRG

```
* VIDREP.PRG V1.00 By Derek Fountain
CLS
SELECT 1
CREATE REP FROM REP.DEF
APPEND BLANK
LET 1:LINE$="TAPE TITLE/STARRING       TYPE DURATION RATING"
SELECT 2
USE VIDEO
AT 0,28 SAY "Home video library system"
LET filename$="          "
AT 2,20 SAY "Please enter the name for the report file"
AT 6,25 SAY "Leave the name blank to abandon"
AT 4,35 GET filename$
IF filename$="          "
  USE
  SELECT 1
  USE
  DELETE FILE REP
  RELEASE filename$
  RETURN
ENDIF
CLS
AT 0,28 SAY "Home video library system"
AT 3,25 SAY "Creating workfile - Please wait"
DO WHILE EOF()=0
  SELECT 1
  APPEND BLANK
  SELECT 2
* following must be typed on one line
  LET 1:LINE$=" "+2:TAPE_NO$+" "+2:TITLE$+" "+2:TYPE$"    "+2:DURATION$+"
"+2:RATING$
  LET key$=2:REF_NUM$
  LET loc=RECNO()
  USE VIDSTAR INDEX VIDSTAR.NDX
  FIND key$
  DO WHILE 2:REF_NUM$=key$
    SELECT 1
    APPEND BLANK
    SELECT 2
    LET 1:LINE$="     "+2:STAR$
    SKIP
  ENDDO
  SELECT 1
  APPEND BLANK
  SELECT 2
  USE VIDEO
  GO loc
  SKIP
ENDDO
AT 3,23 SAY "Creating report file - Please wait"
USE
SELECT 1
GO TOP
COPY TO &filename$ PD
USE
AT 3,23 SAY "Report file created - Press any key"
WAIT
DELETE FILE REP
RETURN
* EOF
```

Wordmongers Stock Control System

This suite was written to demonstrate zBASE' ability to run fully menu driven suites. It was designed with a small shop in mind, where the manager would go round his warehouse, with his Z88 and take a check of all the items he has in stock. The system would then produce a report giving details of how many of product X was in stock and how much it was all worth.

A full manual is not necessary, but a brief description is called for.

Options

Option 1 would be entered, and the journey round the stock room would start. As each product is encountered it's unique code is entered at the prompt. If the code is not to hand, entering a code of a single question mark will allow input of a description of the product. Searching by code is much faster as it uses zBASE's indexing system. Assuming the product is successfully found in the database, the relevant details will be displayed and the number of cases in stock will be requested, followed by the number of individual units. These numbers are used to calculate the total number in stock, and their value. The program then loops round for another code number.

Option 2 is fairly straight forward. It allows the examination and amendment of the products details file.

Option 3 is almost the same as option 2, except it allows for maintenance of the suppliers file.

Option 4 produces a Pipedream plain text file, including all details held in the products file.

Option 5 produces a Pipedream plain text file, including all details held in the suppliers file.

Option 6 produces a detailed 'report' using an intermediate file called REP. It is a mixture of both the products file and the suppliers file. This routine demonstrates zBASE's ability to merge databases and produce the exact output required by the user.

Option 7 cleans the files of all deleted record (note the technique for simulating the dBase PACK command), and recreates both indicies.

Option 8 quits from zBASE.

Wordmongers do not claim this suite is a full program, or that it will perform a task in the most effective manner possible. Its primary function is to demonstrate zBASE's ability to drive large systems.

The code is copyrighted by Wordmongers Ltd, but may be copied and used freely by zBASE users, so long as this copyright is publicly acknowledged, where the code is used.

Here is a list of the files required to run the system. The files can be obtained by sending a blank 32k eprom to us or downloading the zip file.

File Name	Bytes	Description
REP.DEF	128	Definition of file used in stock report
SCFETCH.EXE	512	CLI File to extract these files from EPROM
SCOUTPRO.PRG	1280	Outputs the Product file in PD format
SCOUTSUP.PRG	1280	Outputs the Supplier file in PD format
SCPACK.PRG	768	Cleans, Packs & Reindexs the Databases
SCPROD	384	The Product database
SCPROD.DEF	128	The Definition of the Product database
SCPROD.NDX	128	The index for the Product database
SCSIAM.PRG	1408	Amend Product
SCSIBACK.PRG	640	Skip back one product
SCSIDELE.PRG	512	Delete a product
SCSIFIND.PRG	640	Find a product
SCSIINP.PRG	1920	Input a product

SCSIMAIN.PRG	2176	Products file maintenance menu
SCSINEXT.PRG	384	Skip forward a product
SCSIRECA.PRG	512	Recall a deleted product
SCSTART.PRG	1152	The main menu
SCSTKREP.PRG	2944	The stock report
SCSTOCK.PRG	1920	Amend stock numbers
SCSUAM.PRG	1408	Amend suppliers
SCSUFIND.PRG	768	Find a supplier
SCSUINP.PRG	2048	Input a supplier
SCSUMAIN.PRG	2432	Suppliers file maintenance menu
SCSUPP	384	The Suppliers Database
SCSUPP.NDX	128	The index file for the supplier file
SCSUPP.DEF	128	The definition of the supplier file
ZBDEMO.DOC	4400	This file
ZBRUN	256	The Autoexec file

Program listings

REP.DEF

```
* REP.DEF
LINES,115
```

SCOUTSUP.PRG

```
* SCOUTSUP.PRG V1.01 BY DEREK FOUNTAIN
* OUTPUT SUPPLIERS FILE TO A PD FILE
DO WHILE 1=1
  CLS
  AT 0,30 SAY "OUTPUT SUPPLIER FILE"
  AT 2,20 SAY "Please enter the name of the output file"
  LET filenam$="                    "
  AT 6,25 SAY "Leave the name blank to abandon"
  AT 4,30 GET filenam$
  IF filenam$=" "
    RELEASE filenam$,over$
    RETURN
  ENDIF
  AT 2,20 SAY "                                        "
  AT 4,30 SAY "                    "
  AT 6,25 SAY "                               "
  LET over$="Y"
  IF FILE(filenam$)=1
    AT 3,20 SAY "File exists - Overwrite it? (Y/N) >>>"
    LET over$=" "
    DO WHILE over$=" "
      AT 3,59 GET over$
      IF WHERE(over$,"YyNn")=0
        LET over$=" "
      ENDIF
    ENDDO
  ENDIF
  IF UPPER(over$)="Y"
    AT 3,20 SAY "Copying data to Pipedream file "+filenam$
    USE SCSUPP
    COPY TO &filenam$ PD
    AT 3,20 SAY "                                        "
    AT 3,11 SAY "Process completed - The file MUST be loaded as plain text"
    AT 5,27 SAY "Press any key to continue"
    WAIT
    RELEASE filenam$,over$
    RETURN
  ENDIF
ENDDO
```

SCPACK.PRG

```
* SCPACK.PRG V1.02 BY DEREK FOUNTAIN
* Same as a dBase pack - it removes the deleted record
CLS
AT 0,30 SAY "CLEAN FILES ROUTINE"
AT 3,19 SAY "Removing deleted records may take some time"
AT 5,20 SAY "Confirm you want to proceed (Y/N) >>>"
LET confirm$=" "
AT 5,58 GET confirm$
IF WHERE(confirm$,"Nn")<>0
  RELEASE confirm$
  RETURN
ENDIF
CLS
AT 3,26 SAY "Cleaning files - Please wait"
USE SCPROD
COPY TO TEMP FOR DELETED()=0
USE
DELETE FILE SCPROD
RENAME TEMP TO SCPROD
```

```
USE SCSUPP
COPY TO TEMP FOR DELETED()=0
USE
DELETE FILE SCSUPP
RENAME TEMP TO SCSUPP
USE SCPROD
RELEASE confirm$
RETURN
```

SCPROD.DEF

```
* SCPROD.DEF
CODE$,6
DESC$,20
SUPP$,3
SELL_AT
BUY_AT
CASE
VALUE
MINIMUM
STOCK
LUPDATE$,10
BUGFIX$,10
```

SCSIAM.PRG

```
* SCSIAM.PRG V1.03 BY DEREK FOUNTAIN
* Amend a products details - Called from SCSIMAIN.PRG
* Following at say must be typed on one line - it is 80 spaces.
AT 7,0 SAY "                                        "
        +"                                        "
* The initial data goes into mvars
LET mcode$=1:CODE$
LET mdesc$=1:DESC$
LET msupp$=1:SUPP$
LET msell_at=1:SELL_AT
LET mbuy_at=1:BUY_AT
LET mcase=1:CASE
LET mminimum=1:MINIMUM
* Loop while data is not confirmed
LET confirm$="N"
DO WHILE UPPER(confirm$)="N"
  AT 7,25 SAY "Amend each field, one at a time"
  * Get each field
  AT 2, 5 GET mcode$
  AT 2,35 GET mdesc$
  AT 2,74 GET msupp$
  AT 4, 8 GET msell_at
  AT 4,26 GET mbuy_at
  AT 4,45 GET mcase
  AT 4,71 GET mminimum
* Get confirmation of data
  AT 7,22 SAY "Please confirm this data (Y/N) >>>"
  LET confirm$=" "
  DO WHILE confirm$=" "
    AT 7,57 GET confirm$
    IF WHERE(confirm$,"YNyn")=0
      LET confirm$=" "
    ENDIF
  ENDDO
  AT 7,22 SAY "                                    "
ENDDO           * Loop back if not confirmed
* Got confirmed data in mvars, put them over the old data
LET 1:CODE$=mcode$
LET 1:DESC$=mdesc$
LET 1:SUPP$=msupp$
LET 1:SELL_AT=msell_at
LET 1:BUY_AT=mbuy_at
LET 1:CASE=mcase
LET 1:MINIMUM=mminimum
LET 1:LUPDATE$=today$
RELEASE mcode$,mdesc$,msupp$,msell_at,mbuy_at,mcase,mminimum,confirm$
RETURN
```

SCSIBACK.PRG

```
* SCSIBACK.PRG V1.00 BY DEREK FOUNTAIN
* Moves back a record in the database
* There is a slight problem with skipping backwards. If the
* user tries to skip back past the first record, the record
* pointer sticks at 1. A check is put in the code to detect this
* Beginning Of File flag is set to 0
LET bof=0
IF RECNO()=1
* We are at the beginning of file, so flag it
  LET bof=1
ENDIF
SKIP -1
* If we were at the begining of file, move the other end
IF bof=1
  AT 7,18 SAY "Top of file - Moving to bottom. Press any key"
  WAIT
  GO BOTTOM              * Go to bottom of file if we have
ENDIF
RELEASE bof
RETURN
```

SCSIDELE.PRG

```
* SCSIDELE.PRG V1.01 BY DEREK FOUNTAIN
* Deletes the current record in the database
IF DELETED()=0           * Not already deleted
  AT 7,12 SAY "Please confirm this record is to be deleted (Y/N) >>>"
  LET confirm$=" "
  DO WHILE confirm$=" "
```

```
    AT 7,67 GET confirm$
    IF WHERE(confirm$,"YNyn")=0
      LET confirm$=" "
    ENDIF
  ENDDO
  IF UPPER(confirm$)="Y"
    IF calling$="SCSI"
      LET 1:LUPDATE$=today$
    ENDIF
    DELETE RECORD
  ENDIF
ENDIF
RELEASE confirm$
RETURN
```

SCSIFIND.PRG

```
* SCSIFIND.PRG V1.00 BY DEREK FOUNTAIN
* Finds a specified record
AT 2, 5 SAY "      "
AT 2,35 SAY "                    "
AT 2,74 SAY "   "
AT 4, 8 SAY "        "
AT 4,26 SAY "        "
AT 4,45 SAY "        "
AT 4,71 SAY "        "
LET mcode$="      "
AT 2,5  GET mcode$
IF mcode$<>" "
  AT 2,5 SAY "SEARCHING"
  LET loc=RECNO()
  LOCATE FOR 1:CODE$=mcode$
  AT 2,5 SAY "         "
  IF EOF()=1
    AT 2,5 SAY "NO FIND  "
    LET count=10
    DO WHILE count>0
      LET count=count-1
    ENDDO
    AT 2,5 SAY "         "
    GO loc
  ENDIF
ENDIF
RELEASE mcode$,loc,count
RETURN
```

SCSIINP.PRG

```
* SCSIINP.PRG V1.03 BY DEREK FOUNTAIN
* Input of new products details - Called from SCSIMAIN.PRG
AT 7,0 SAY "                                                                              "
* The initial data goes into mvars
LET mcode$="      "
LET mdesc$="                    "
LET msupp$="   "
LET msell_at=0.00
LET mbuy_at=0.00
LET mcase=0.00
LET mminimum=0.00
* Loop while data is not confirmed
LET confirm$="N"
DO WHILE UPPER(confirm$)="N"
  AT 0,0 SAY "                    "
  AT 0,56 SAY "                    "
  AT 2, 5 SAY "      "
  AT 2,35 SAY "                "
  AT 2,74 SAY "   "
  AT 4, 8 SAY "        "
  AT 4,26 SAY "        "
  AT 4,45 SAY "        "
  AT 4,71 SAY "         "
  AT 7,25 SAY "Enter each field, one at a time"
* Get each field
  AT 2, 5 GET mcode$
  IF mcode$="      "
    RELEASE mcode$,mdesc$,msupp$,msell_at,mbuy_at,mcase,mminimum,confirm$
    RETURN
  ENDIF
  AT 2,35 GET mdesc$
  AT 2,74 GET msupp$
  AT 4, 8 GET msell_at
  AT 4,26 GET mbuy_at
  AT 4,45 GET mcase
  AT 4,71 GET mminimum
* Get confirmation of data
  AT 7,21 SAY "Please confirm this data (Y/N/Q) >>>"
  LET confirm$=" "
  DO WHILE confirm$=" "
    AT 7,58 GET confirm$
    IF WHERE(confirm$,"YNQynq")=0
      LET confirm$=" "
    ENDIF
  ENDDO
  IF UPPER(confirm$)="Q"
    RELEASE mcode$,mdesc$,msupp$,msell_at,mbuy_at,mcase,mminimum,confirm$
    RETURN
  ENDIF
  AT 7,21 SAY "                                    "
ENDDO            * Loop back if not confirmed
* Got confirmed data in mvars, put them into a new record
APPEND BLANK
LET 1:CODE$=mcode$
LET 1:DESC$=mdesc$
LET 1:SUPP$=msupp$
LET 1:SELL_AT=msell_at
LET 1:BUY_AT=mbuy_at
LET 1:CASE=mcase
LET 1:MINIMUM=mminimum
LET 1:LUPDATE$=today$
RELEASE mcode$,mdesc$,msupp$,msell_at,mbuy_at,mcase,mminimum,confirm$
RETURN
```

SCSIMAIN.PRG

```
* SCSIMAIN.PRG V1.09 BY DEREK FOUNTAIN
* Maintenance of products file - Called by SCSTART.PRG
CLS
USE SCPROD
* Set up screen
AT 0,26 SAY "PRODUCT DATA FILE MAINTENANCE"
AT 2,0 SAY "CODE:"
AT 2,30 SAY "ITEM:"
AT 2,60 SAY "SUPPLIER CODE:"
AT 4,0 SAY "SELL AT:"
AT 4,19 SAY "BUY AT:"
AT 4,40 SAY "CASE:"
AT 4,57 SAY "MINIMUM STOCK:"
* For the benefit of the delete routines...
LET calling$="SCSI"
* Infinite loop, keep getting choices
DO WHILE 1=1
* Print up the current record
  AT 0,65 SAY "DATED:"
  AT 2, 5 SAY "     "
  AT 2,35 SAY "                    "
  AT 2,74 SAY "   "
  AT 4, 8 SAY "        "
  AT 4,26 SAY "        "
  AT 4,45 SAY "        "
  AT 4,71 SAY "        "
  AT 0,71 SAY 1:LUPDATE$
  AT 2,5 SAY 1:CODE$
  AT 2,35 SAY 1:DESC$
  AT 2,74 SAY 1:SUPP$
  AT 4,8 SAY 1:SELL_AT
  AT 4,26 SAY 1:BUY_AT
  AT 4,45 SAY 1:CASE
  AT 4,71 SAY 1:MINIMUM
  IF DELETED()=1
    AT 0,0 SAY "DELETED RECORD"
    AT 7,10 SAY "(N)ext,(B)ack,(F)ind,(R)ecall,(I)nput,(A)mend or (Q)uit >>>"
  ELSE
    AT 0,0 SAY "                    "
    AT 7,10 SAY "(N)ext,(B)ack,(F)ind,(D)elete,(I)nput,(A)mend or (Q)uit >>>"
  ENDIF
* Get a valid choice
  LET choice$=" "
  DO WHILE choice$=" "
    AT 7,70 GET choice$
    IF WHERE(UPPER(choice$),"NBFDRIAQ")=0
      LET choice$=" "
    ENDIF
  ENDDO
* Clear prompt form bottom line
  AT 7,10 SAY "                                                  "
  IF UPPER(choice$)="Q"          * Option was to quit
    RELEASE calling$
    RETURN
  ENDIF
  IF UPPER(choice$)="N"          * Option was Next record
    DO SCSINEXT.PRG
  ENDIF
  IF UPPER(choice$)="B"          * Choice was to move Back a record
    DO SCSIBACK.PRG
  ENDIF
```

```
 IF UPPER(choice$)="I"          * Choice was to input a new record
   DO SCSIINP.PRG
 ENDIF
 IF UPPER(choice$)="A"          * Choice was to amend current record
   DO SCSIAM.PRG
 ENDIF
 IF UPPER(choice$)="F"          * Choice was Find
   DO SCSIFIND.PRG
 ENDIF
 IF UPPER(choice$)="D"          * Choice was delete
   DO SCSIDELE.PRG
 ENDIF
 IF UPPER(choice$)="R"          * Choice was recall
   DO SCSIRECA.PRG
 ENDIF
ENDDO
```

SCSINEXT.PRG

```
* SCSINEXT.PRG V1.00 BY DEREK FOUNTAIN
* Skips to next record in database
SKIP
IF EOF()=1               * Check we haven't skipped to far
  AT 7,19 SAY "End of file - Moving to top. Press any key"
  WAIT
  GO TOP                 * Back to the top if we have
ENDIF
RETURN
```

SCSIRECA.PRG

```
* SCSIRECA.PRG V1.01 BY DEREK FOUNTAIN
* Recalls the current record in the database
IF DELETED()=1              * Must be deleted already
  AT 7,12 SAY "Please confirm this record is to be recalled (Y/N) >>>"
  LET confirm$=" "
  DO WHILE confirm$=" "
    AT 7,68 GET confirm$
    IF WHERE(confirm$,"YNyn")=0
      LET confirm$=" "
    ENDIF
  ENDDO
  IF UPPER(confirm$)="Y"
    IF calling$="SCSI"
      LET 1:LUPDATE$=today$
    ENDIF
    RECALL RECORD
  ENDIF
ENDIF
RELEASE confirm$
RETURN
```

SCSTART.PRG

```
* SCSTART.PRG V1.04 BY DEREK FOUNTAIN
CLS
AT 0,24 SAY "WORDMONGERS STOCK CONTROL SYSTEM"
LET today$="        "
AT 3,21 SAY "Please enter todays date >>>"
AT 3,50 GET today$
DO WHILE 1=1
  CLS
  AT 0,24 SAY "WORDMONGERS STOCK CONTROL SYSTEM"
  AT 2,16 SAY "1) STOCK CHECK             5) SUPPLIERS FILE REPORT"
  AT 3,16 SAY "2) PRODUCTS - ENTER/AMEND    6) STOCK VALUATION"
  AT 4,16 SAY "3) SUPPLIERS - ENTER/AMEND   7) CLEAN FILES"
  AT 5,16 SAY "4) PRODUCTS FILE REPORT      8) EXIT SYSTEM"
  AT 7,30 SAY "PLEASE SELECT >>>"
  LET choice$=" "
  DO WHILE choice$=" "
    AT 7,48 GET choice$
    IF WHERE(choice$,"12345678")=0
      LET choice$=" "
    ENDIF
  ENDDO
  IF choice$="1"
    DO SCSTOCK.PRG
  ENDIF
  IF choice$="2"
    DO SCSIMAIN.PRG
  ENDIF
  IF choice$="3"
    DO SCSUMAIN.PRG
  ENDIF
  IF choice$="4"
    DO SCOUTPRO.PRG
  ENDIF
  IF choice$="5"
    DO SCOUTSUP.PRG
  ENDIF
  IF choice$="6"
    DO SCSTKREP.PRG
  ENDIF
  IF choice$="7"
    DO SCPACK.PRG
  ENDIF
  IF choice$="8"
    AT 5,0 SAY "                4) PR"
    QUIT
  ENDIF
ENDDO
```

SCSTKREP.PRG

```
* SCSTKREP.PRG V1.01 BY DEREK FOUNTAIN
* Outputs a stock report to the specified file
* Uses a new file with one very large field to dump the data
CLS
AT 0,34 SAY "STOCK REPORT"
AT 2,19 SAY "This report involves generating a work file"
AT 3,25 SAY "This process may take some time"
AT 5,8 SAY "Please confirm you wish to generate the report file (Y/N) >>>"
LET confirm$=" "
AT 5,70 GET confirm$
IF UPPER(confirm$)="N"
  RELEASE confirm$
  RETURN
ENDIF
CLS
AT 0,34 SAY "STOCK REPORT"
AT 3,24 SAY "Creating work file - Please wait"
SELECT 2
CREATE REP FROM REP.DEF
SELECT 1
USE SCPROD
GO TOP
DO WHILE EOF()=0
  SELECT 2
  APPEND BLANK
  SELECT 1
  LET mline$=1:CODE$+" "+1:DESC$+" "
  LET msupp$=1:SUPP$
  LET loc=RECNO()
  USE SCSUPP
  LOCATE FOR 1:SUPP$=msupp$
  LET msupp$=1:NAME$
  IF EOF()=1
    LET msupp$="Supplier not on file      "
  ENDIF
  USE SCPROD
  GO loc
  LET var1=1:BUY_AT
  LET lvar1=LEN("&var1")
  LET var2=1:SELL_AT
  LET lvar2=LEN("&var2")
  LET var3=1:VALUE
  LET lvar3=LEN("&var3")
  LET var4=1:STOCK
  LET lvar4=LEN("&var4")
  LET lmsupp=LEN(TRIM(msupp$))
  LET 2:LINE$=mline$+" "+msupp$
  LET 2:LINE$=TRIM(2:LINE$)+
      SUBSTR("                         ",1,25-lmsupp)+" &var1"
  LET 2:LINE$=TRIM(2:LINE$)+SUBSTR("          ",1,10-lvar1)+" &var2"
  LET 2:LINE$=TRIM(2:LINE$)+SUBSTR("          ",1,10-lvar2)+" &var3"
  LET 2:LINE$=TRIM(2:LINE$)+SUBSTR("          ",1,10-lvar3)+" &var4"
  LET 2:LINE$=TRIM(2:LINE$)+SUBSTR("          ",1,10-lvar4)+" "+1:LUPDATE$
  SKIP
ENDDO
SELECT 2
GO TOP
AT 3,24 SAY "                              "
DO WHILE 1=1
  AT 2,20 SAY "Please enter the name of the output file"
  LET filenam$="                "
  AT 6,25 SAY "Leave the name blank to abandon"
  AT 4,30 GET filenam$
  IF filenam$=" "
    USE
    DELETE FILE REP
    SELECT 1
    RELEASE confirm$,msupp$,filenam$,over$,mline$,loc
    RELEASE var1,var2,var3,var4,lvar1,lvar2,lvar3,lvar4
    RETURN
  ENDIF
  AT 2,20 SAY "                                        "
  AT 4,30 SAY "                    "
  AT 6,25 SAY "                              "
  LET over$="Y"
  IF FILE(filenam$)=1
    AT 3,20 SAY "File exists - Overwrite it? (Y/N) >>>"
    LET over$=" "
    DO WHILE over$=" "
      AT 3,59 GET over$
      IF WHERE(over$,"YyNn")=0
        LET over$=" "
      ENDIF
    ENDDO
  ENDIF
  AT 3,20 SAY "                                        "
  IF UPPER(over$)="Y"
    AT 3,20 SAY "Copying data to Pipedream file "+filenam$
    COPY TO &filenam$ PD
    AT 3,20 SAY "                                        "
    AT 3,11 SAY "Process completed - The file MUST be loaded as plain text"
    AT 5,27 SAY "Press any key to continue"
    WAIT
    USE
    DELETE FILE REP
    SELECT 1
    RELEASE confirm$,msupp$,filenam$,over$,mline$,loc
    RELEASE var1,var2,var3,var4,lvar1,lvar2,lvar3,lvar4
    RETURN
    ENDIF
  ENDDO
```

SCSTOCK.PRG

```
* SCSTOCK.PRG V1.05 BY DEREK FOUNTAIN
* THIS ROUTINE ALLOWS UPDATE OF THE NUMBER OF A SELECTED ITEM
* IN STOCK
CLS
AT 0,30 SAY "STOCK UPDATE ROUTINE"
USE SCPROD          * Products database
DO WHILE 1=1        * Infinite loop
  LET mcode$="     "
  LET mdesc$="                    "
  AT 2,0 SAY "CODE:     "
  AT 2,30 SAY "ITEM:                    "
  AT 2,60 SAY "SUPPLIER CODE:   "
  AT 4,0 SAY "SELLING PRICE:        "
  AT 4,30 SAY "BUYING PRICE:        "
  AT 4,60 SAY "NO PER CASE:        "
  AT 6,0 SAY "CURRENT STOCK:        "
  AT 6,30 SAY "MINIMUM STOCK:        "
  AT 6,60 SAY "                    "
  AT 2,5 GET mcode$
  IF mcode$="     "
    RELEASE mcode$,p,mcases,msingles
    RETURN
  ENDIF
  IF mcode$="?"
    AT 2,5 SAY "     "
    AT 2,35 GET mdesc$
    IF mdesc$=" "
      RETURN
    ENDIF
    LOCATE FOR UPPER(1:DESC$)=UPPER(TRIM(mdesc$))
```

```
ELSE
  LOCATE FOR 1:CODE$=mcode$
ENDIF
IF EOF()=1
  AT 2,5 SAY "NO FIND"
  LET p=30
  DO WHILE p>0
    LET p=p-1
  ENDDO
  AT 2,35 SAY "                    "
  AT 2,5 SAY "       "
  LET mcode$="      "
ENDIF
IF mcode$<>"      "
  AT 2,5 SAY 1:CODE$
  AT 2,35 SAY 1:DESC$
  AT 2,74 SAY 1:SUPP$
  AT 4,15 SAY 1:SELL_AT
  AT 4,43 SAY 1:BUY_AT
  AT 4,72 SAY 1:CASE
  AT 6,14 SAY 1:STOCK
  AT 6,44 SAY 1:MINIMUM
  IF mcode$="?"
    LET confirm$="Y"
    AT 7,13 SAY "Please confirm this is the correct product (Y/N) >>>"
    AT 7,66 GET confirm$
    AT 7,13 SAY "                                        "
    IF UPPER(confirm$)="N"
      ENDDO
    ENDIF
  ENDIF
  LET mcases=0
  LET msingles=0
  AT 6,62 SAY "CASES?:"
  AT 6,69 GET mcases
```

```
    AT 6,60 SAY "SINGLES?:"
    AT 6,69 SAY "        "
    AT 6,69 GET msingles
    LET 1:STOCK=mcases*1:CASE+msingles
    LET 1:VALUE=1:STOCK*1:SELL_AT
    LET 1:LUPDATE$=today$
  ENDIF
ENDDO
```

SCSUAM.PRG

```
* SCSUAM.PRG V1.01 BY DEREK FOUNTAIN
* Amend of suppliers details - Called from SCSUMAIN.PRG
AT 7,0 SAY "                                                              "
* The initial data goes into mvars
LET msupp$=1:SUPP$
LET mname$=1:NAME$
LET mcontact$=1:CONTACT$
LET mphone$=1:PHONE$
LET madd1$=1:ADD1$
LET madd2$=1:ADD2$
LET madd3$=1:ADD3$
LET madd4$=1:ADD4$
* Loop while data is not confirmed
LET confirm$="N"
DO WHILE UPPER(confirm$)="N"
  AT 7,25 SAY "Amend each field, one at a time"
  * Get each field
  AT 2,12 GET mname$
  AT 3,12 GET msupp$
  AT 4,12 GET mcontact$
  AT 5,12 GET mphone$
  AT 2,48 GET madd1$
  AT 3,48 GET madd2$
  AT 4,48 GET madd3$
```

```
 AT 5,48 GET madd4$
* Get confirmation of data
 AT 7,22 SAY "Please confirm this data (Y/N) >>>"
 LET confirm$=" "
 DO WHILE confirm$=" "
   AT 7,57 GET confirm$
   IF WHERE(confirm$,"YNyn")=0
     LET confirm$=" "
   ENDIF
 ENDDO
 AT 7,21 SAY "                                     "
ENDDO           * Loop back if not confirmed
* Got confirmed data in mvars, put them over the old data
LET 1:SUPP$=msupp$
LET 1:NAME$=mname$
LET 1:CONTACT$=mcontact$
LET 1:PHONE$=mphone$
LET 1:ADD1$=madd1$
LET 1:ADD2$=madd2$
LET 1:ADD3$=madd3$
LET 1:ADD4$=madd4$
RELEASE msupp$,mname$,mcontact$,mphone$,madd1$,madd2$,madd3$,madd4$,confirm$
RETURN
```

SCSUFIND.PRG

```
* SCSUFIND.PRG V1.01 BY DEREK FOUNTAIN
* Finds a specified record
AT 2,11 SAY "                    "
AT 3,11 SAY "    "
AT 4,11 SAY "                    "
AT 5,11 SAY "            "
AT 2,48 SAY "                    "
AT 3,48 SAY "                    "
AT 4,48 SAY "                    "
AT 5,48 SAY "                    "
LET msupp$="   "
AT 3,12 GET msupp$
IF msupp$<>" "
  AT 3,12 SAY "SEARCHING"
  LET loc=RECNO()
  LOCATE FOR 1:SUPP$=msupp$
  AT 3,12 SAY "         "
  IF EOF()=1
    AT 3,12 SAY "NO FIND  "
    LET count=10
    DO WHILE count>0
      LET count=count-1
    ENDDO
    AT 3,12 SAY "         "
    GO loc
  ENDIF
ENDIF
RELEASE msupp$,loc,count
RETURN
```

SCSUINP.PRG

```
* SCSUINP.PRG V1.01 BY DEREK FOUNTAIN
* Input of new suppliers details - Called from SCSUMAIN.PRG
AT 7,0 SAY "                                                  "
* The initial data goes into mvars
LET msupp$="   "
LET mname$="                    "
LET mcontact$="                    "
LET mphone$="              "
LET madd1$="                    "
LET madd2$="                    "
LET madd3$="                    "
LET madd4$="                    "
```

```
* Loop while data is not confirmed
LET confirm$="N"
DO WHILE UPPER(confirm$)="N"
  AT 2,11 SAY "                         "
  AT 3,11 SAY "     "
  AT 4,11 SAY "                         "
  AT 5,11 SAY "               "
  AT 2,48 SAY "                         "
  AT 3,48 SAY "                         "
  AT 4,48 SAY "                         "
  AT 5,48 SAY "                         "
  AT 7,25 SAY "Enter each field, one at a time"
  * Get each field
  AT 2,12 GET mname$
  IF mname$="      "
    RELEASE msupp$,mname$,mcontact$,mphone$
    RELEASE madd1$,madd2$,madd3$,madd4$,confirm$
    RETURN
  ENDIF
  AT 3,12 GET msupp$
  AT 4,12 GET mcontact$
  AT 5,12 GET mphone$
  AT 2,48 GET madd1$
  AT 3,48 GET madd2$
  AT 4,48 GET madd3$
  AT 5,48 GET madd4$
* Get confirmation of data
  AT 7,21 SAY "Please confirm this data (Y/N/Q) >>>"
  LET confirm$=" "
  DO WHILE confirm$=" "
    AT 7,58 GET confirm$
    IF WHERE(confirm$,"YNQynq")=0
      LET confirm$=" "
    ENDIF
  ENDDO
  IF UPPER(confirm$)="Q"
    RELEASE msupp$,mname$,mcontact$,mphone$
    RELEASE madd1$,madd2$,madd3$,madd4$,confirm$
    RETURN
  ENDIF
  AT 7,21 SAY "                              "
ENDDO             * Loop back if not confirmed
* Got confirmed data in mvars, put them into a new record
APPEND BLANK
LET 1:SUPP$=msupp$
LET 1:NAME$=mname$
LET 1:CONTACT$=mcontact$
LET 1:PHONE$=mphone$
LET 1:ADD1$=madd1$
LET 1:ADD2$=madd2$
LET 1:ADD3$=madd3$
LET 1:ADD4$=madd4$
RELEASE msupp$,mname$,mcontact$,mphone$,madd1$,madd2$,madd3$,madd4$,confirm$
RETURN
```

SCSUMAIN.PRG

```
* SCSUMAIN.PRG V1.02 BY DEREK FOUNTAIN
* Maintenance of suppliers file - Called by SCSTART.PRG
CLS
USE SCSUPP
* Set up screen
AT 0,25 SAY "SUPPLIERS DATA FILE MAINTENANCE"
AT 2,2 SAY "SUPPLIER:"
AT 3,2 SAY "CODE    :"
AT 4,2 SAY "CONTACT :"
AT 5,2 SAY "PHONE   :"
AT 2,40 SAY "ADDRESS:"
* Infinite loop, keep getting choices
LET calling$="SCSU"
```

```
DO WHILE 1=1
  * Print up the current record
  AT 2,11 SAY "                          "
  AT 3,11 SAY "     "
  AT 4,11 SAY "                          "
  AT 5,11 SAY "               "
  AT 2,48 SAY "                          "
  AT 3,48 SAY "                          "
  AT 4,48 SAY "                          "
  AT 5,48 SAY "                          "
  AT 2,12 SAY 1:NAME$
  AT 3,12 SAY 1:SUPP$
  AT 4,12 SAY 1:CONTACT$
  AT 5,12 SAY 1:PHONE$
  AT 2,48 SAY 1:ADD1$
  AT 3,48 SAY 1:ADD2$
  AT 4,48 SAY 1:ADD3$
  AT 5,48 SAY 1:ADD4$
  IF DELETED()=1
    AT 0,0 SAY "DELETED RECORD"
    AT 7,10 SAY "(N)ext,(B)ack,(F)ind,(R)ecall,(I)nput,(A)mend or (Q)uit >>>"
  ELSE
    AT 0,0 SAY "                          "
    AT 7,10 SAY "(N)ext,(B)ack,(F)ind,(D)elete,(I)nput,(A)mend or (Q)uit >>>"
  ENDIF
* Get a valid choice
  LET choice$=" "
  DO WHILE choice$=" "
    AT 7,70 GET choice$
    IF WHERE(UPPER(choice$),"NBFDRIAQ")=0
      LET choice$=" "
    ENDIF
  ENDDO
* Clear prompt form bottom line
  AT 7,10 SAY "                                                  "
```

```
 IF UPPER(choice$)="Q"          * Option was to quit
  RELEASE calling$
  RETURN
 ENDIF
 IF UPPER(choice$)="N"          * Option was Next record
  DO SCSINEXT.PRG               * This is the same routine as in
 ENDIF                  * the stock file maintenance
 IF UPPER(choice$)="B"          * Choice was to move Back a record
  DO SCSIBACK.PRG               * Ditto
 ENDIF
 IF UPPER(choice$)="I"         * Choice was to input a new record
  DO SCSUINP.PRG
 ENDIF
 IF UPPER(choice$)="A"          * Choice was to amend current record
  DO SCSUAM.PRG
 ENDIF
 IF UPPER(choice$)="F"          * Choice was Find
  DO SCSUFIND.PRG
 ENDIF
 IF UPPER(choice$)="D"          * Choice was delete
  DO SCSIDELE.PRG               * Same routine as used in the stock
 ENDIF                  * item maintenance
 IF UPPER(choice$)="R"          * Choice was recall
  DO SCSIRECA.PRG               * Ditto
 ENDIF
ENDDO
```

SCSUPP.DEF

```
* SCSUPP.DEF
SUPP$,3
NAME$,25
ADD1$,25
ADD2$,25
ADD3$,25
ADD4$,25
PHONE$,15
CONTACT$,25
```

zBASE Pocket Ref. Guide

CONVENTIONS

Lowercase	operator input, usually enclosed in < > brackets.
UPPERCASE	specific zBASE commands or command words.
[....]	Optional parts of commands.
<...>	Operator specified input.
<exp>...	An expression which can result in either a number or a string. e.g. 5+5, "FRED"+"BLOGGS", a$+"MUMMY".
<var> ...	A variable, can mean either mvar or fvar.
<mvar>...	A memory variable, not stored in a database, but in a large buffer in RAM. An mvar is defined as a string if its name ends with a '$', otherwise, it is defined as a number.
<fvar>...	A field variable, permanently stored in a database. Fields must start with either a 1: or 2: label, depending on the database they are to be taken from. If the label is missing the field will be taken as a mvar.
<cond>...	A condition which returns the result either TRUE or FALSE. e.g. 10=10 is TRUE, 10=6 is FALSE, "FRED"="BLOGGS" is FALSE. A single number 0 is evaluated as FALSE, any other number is TRUE.

KEYWORDS

Only the first 4 characters of a keyword are significant.

zBASE commands

*****
adds comments to a command file
? [<exp>]
displays the value of an expression.
[<exp>]
sends the value of <exp> expression through the serial port to the printer.
AT <co-ordinates> SAY <exp>
displays the value of the <exp> expression at the specified co-ordinates.
AT <co-ordinates> GET <var>
formats fields on screen for operator input.
APPEND BLANK
adds a blank record to the database in use
APPEND FROM <filename> [PD]/[DELIMITED] [FOR <exp>]]
adds new records to the current database from another database or Pipedream file.
CLS

clears the screen.
CONTINUE
extension to LOCATE command to move to next match.
COPY TO <filename> [FOR <cond>]
creates new database from current one with optional conditional selection.
COPY TO <filename> [PD] {DELIMITED] [FOR<cond>]
copies data from current file to new format with optional conditional pull.
COPY TO <filename> STRUCTURE [DELIMITED]
creates a database file with same structure as current file, or a Pipedream file of the structure.
CREATE <file1> FROM <file2>
creates zBASE file called 'file1' from Pipedream file 'file2' of format FIELD_NAME, with $ if a string field COMMA, WIDTH if its string field.
DELETE RECORD
marks current record for deletion.
DELETE FILE
removes selected file from directory.
DISPLAY<[STRUCTURE]/[MEMORY]/[STATUS]>
shows on screen the selected option related to current use of database, memory variables and files respectively.
DISPLAY <[ALL] / [FOR <cond>]> <[FIELDS field,field,field]>

shows data from current file in use [ALL] - shows seven records before pausing.
DO <command file>
runs a command file.
DO WHILE <cond> - ENDDO
runs the commands enlosed in loop as long as <cond> is TRUE.
FIND <exp>
searches fo key field match in indexed file.
GO [<exp> / <BOTTOM> / <TOP>]
moves record pointer to <exp>th record or top/bottom.
IF <cond> - ELSE - ENDIF
runs the commands after IF if <cond> is TRUE, otherwise runs commands after ELSE.
INDEX ON <fvar> TO <filename>
creates an index file in order of fvar.
LET <var>=<exp>
establishes a value for a variable.
LIST [ALL]
works as DISPLAY without the pause every 8 lines. [ALL] - shows all records without pausing.
LOCATE FOR <cond>

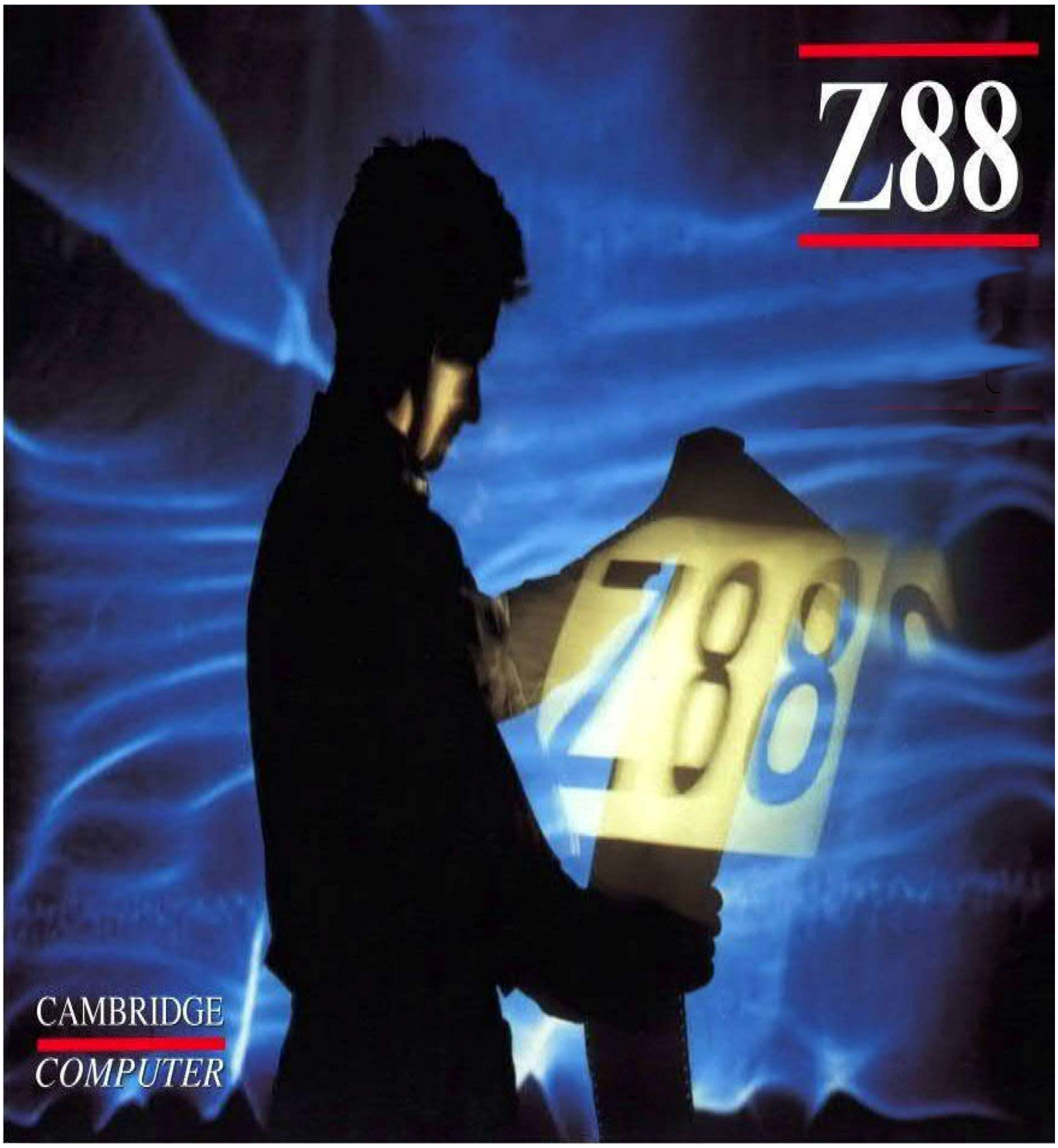
moves record pointer to first record in file for which <cond> is TRUE.
QUIT
closes all files and zBASE application.
RECALL RECORD
removes DELETE mark on a record
RELEASE <mvar>
removes specified memory variables.
RENAME <file1> TO <file2>
changes name of file.
RETURN
stops running current command file and returns control to previous command file or curly prompt.
SELECT <[1 or 2]>
opens selected database area.
SKIP [<exp>]
moves record pointer <exp> records along.
USE [<file>]
closes current file and opens <file> if specified.
USE file INDEX file
opens database in current area with index file.

WAIT
pauses operation until a key is pressed.

zBASE Functions

CHR(<exp>)	returns charater with ASCIIcode exp.
CLI (<exp$)	sends <exp$> to the OZ CLI function for immediate execution.
DATE()	provides current system date.
DELETED()	returns 1 or 0 (TRUE / FALSE) to reflect status of current record.
EOF()	returns 1 or 0 if end of file has been reached or not respectively.
FILE(<exp$>)	responds with TRUE if file defined by exp does exist.
INT(<exp>)	returns integer from exp.
LEN(<exp$>)	shows length of string variable specified.
LOWER(<exp$>)	turns string into lower case.
LTRIM(<exp$>)	removes left hand blanks in string exp.
RAM()	reveals RAM space available on currently selected device.
RECNO()	returns current record number.
SET ECHO	toggles echoing of all commands to screen.
STR	To emulate a **STR** function use &. } LET A=2.03 } LET A$="&A" } ? "-"+a$+"-" -2.03-
SUBSTR(<exp1$>, <exp2>,<exp3>)	extracts the sub-string from exp1$ defined as starting at position exp2, of length exp3.

TIME()	displays system time.
TRIM(<exp$>)	removes trailing blanks.
UPPER(<exp$>)	converts string exp$ to upper case.
VAL(<exp$>)	converts ASCII string to its numeric equivalnet.
WHERE(<exp1$>, <exp2$>)	shows the starting character position for where exp1$ occurs in exp2$.


Z88
Z88
CAMBRIDGE
COMPUTER