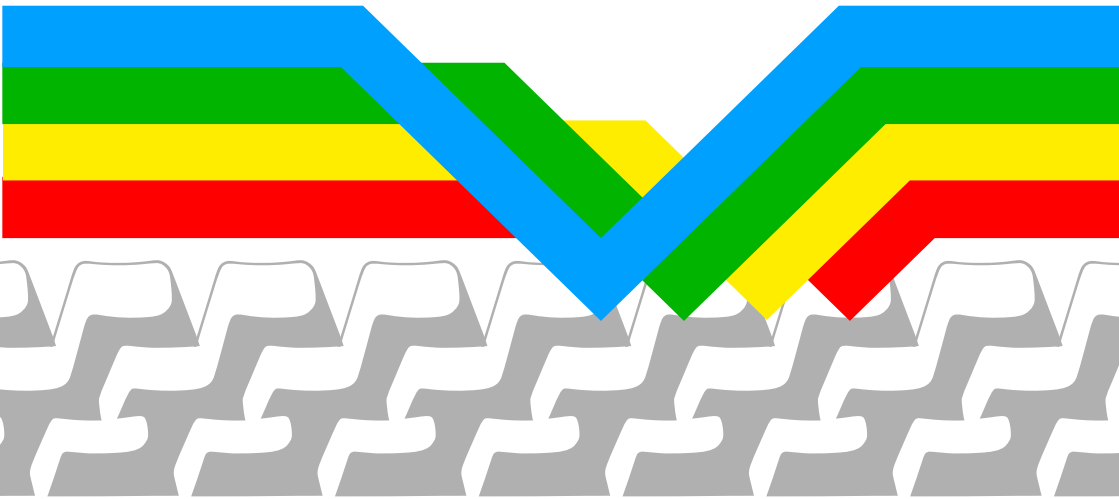
Precision Books

Superbase™ THE BOOK

Dr Bruce Hunt

- *Programming*
- *System Design*
- *Troubleshooting Guide*



SUPERBASE: THE BOOK

A Guide to Database Applications

Dr. Bruce Hunt

Precision Books

1986

First Published 1986 by Precision Software Limited

ISBN 1 85231 000 6

Printed by Repro Workshop Limited, Alton, Hampshire. Mastered on Superscript 128 and Canon LBP-8 A2 Laser Printer.

Precision Software Limited, 6 Park Terrace, Worcester Park, Surrey KT4 7JZ, England.

USA Distribution: Progressive Peripherals & Software, 464 Kalamath Street, Denver, CO 80204.

CONTENTS

Introduction

Since Superbase 64 was first released in 1983 there have been many requests for a "Book of the Program". We have been promising one for almost as long, but every attempt has come to nothing, energy and resources always being needed for some other project. However, at last, here it is, a book which although it cannot lay claim to being definitive, has at least the merit of filling a gap.

I have tried to provide an extension of the Superbase Manual rather than an alternative to it. This means that everyone expecting to benefit from the discussions and examples in the text should really have read the manual first. However, expertise in Superbase is not required: the book caters for all levels of familiarity with the Superbase system, and should help people experimenting at home as much as those trying to set up a serious office system.

There are three main sections to the book and an appendix. Anyone looking for a solution to an immediate problem with Superbase should turn at once to the Troubleshooting appendix. Otherwise, the three sections are arranged to match increasing levels of knowledge of the Superbase system.

Part One, Setting up a System, is for users who have recently acquired Superbase and want to design a database on sound principles, and for those who feel that their existing systems may not be doing the best possible job. The chapters take the reader from the essential process of analysis and design of file structure, with discussions of topics such as index key construction methods, screen design, and database components, through to the core menu options for data entry, database searching, sorting, and output.

In Part Two, The Automated Database, users can learn how to build on the foundation of Part One and construct their own simple Superbase programs. A guide to effective use of the Superbase command line develops into a worked example of a simple program, and advise on how to manage a programmed system. The remaining chapters expand on the topics raised in Part One, showing how to transform simple menu selections into powerful time-saving automatic routines. A final chapter concentrates on Superbase's reporting functions.

Part Three, The Programmed Database, goes further into the mysteries of the Superbase programming language. Three techniques for menu creation are discussed in the context of general screen handling. Then the lengthy Chapter 10 explores many advanced topics, including multi-file applications, reporting refinements such as page numbering, database reorganization, and the 'do/perform' metacommand. The last chapter gives some pointers on how to parameterize a system to gain benefits of increased flexibility and reduced overhead.

Finally, the Troubleshooting appendix lists the known errors in all Superbase versions and provides you with a means of identifying the most common problem areas and how to deal with them.

Introduction

This book owes a great deal to others. My colleagues and friends in Britain and the United States, as well as the 100,000 Superbase users around the world, have contributed greatly to the stock of knowledge on which the book is based. In particular, Simon Tranmer, the designer of Superbase, has helped me to transform technical arcana into plainer truths. And we must all salute the expertise and dedication of the former Superbase Technical Support Manager, Brian Leighfield.

Before you take the plunge, a word of warning. The examples in the book have all been tested, and they work as far as we know. But they are only examples, deliberately simplified for the book, and they are not intended to be copied directly into a working system without modification. You should always expect to have to make a few changes.

PART I

SETTING UP A SYSTEM

CHAPTER 1
FIRST STEPS: DESIGNING FILES AND INDEXES

Have you read the Superbase Manual? In particular, have you worked through the tutorials? If you want to make the most of this section of the book you must have some knowledge of the basics of the Superbase system. I assume that you know, for example, how to load the program, create a work disk, and set up a database for yourself. You should also know how to move between the two main menus and the Select and Maintain submenus, how to select options from the menu, and how to exit from options when you no longer need them. All this is covered in the Superbase manual, and you won't find much guidance at this level of basic operation here. If you don't know how to do any of the things mentioned above, I suggest you go back to the manual and look them up (there are references in either the Table of Contents or the Index). Then come back to this book and read on.

Anybody can set up a Superbase database after a few hours work, enter data into it, and retrieve it for screen display or printed output. But there is a great difference between a system that reflects a first attempt to understand Superbase, which for some people is indeed their very first contact with computers, and a system that incorporates a greater depth of experience. Many people go through the sometimes frustrating process of setting up systems, running them for some weeks, and then having to start again when they realize that they have designed their systems in such a way that many of Superbase's powerful features cannot be used. This is the point at which Precision Software receives many a plea for help. The following pages try to remedy this situation.

My aim is to help you jump over some of the delay in learning what makes a good system. I do this by concentrating on the underlying elements of every system, trying to focus on just the menu options that really matter and how to use them most effectively. Here is a list of the options I'll be discussing, and the actual elements of work they correspond to.

file	Thinking about your data, deciding how to group it on the basis of how you plan to use it, especially how you plan to print it out.
format	Designing an "input screen" so that it's legible and logical; understanding the best ways to use the different types of data field.
enter select add select replace import	Putting record data into the file, amending records already there, and loading up large numbers of records automatically.
select key select n/p/f/l select current	Calling records back to the screen with the "key field", flipping through the file, understanding the idea of the "current record".

select match find	Searching through the file for certain records you want to work with, for example to delete, to sort into a new order, or to print out.
batch	Going through the file making changes to some of the records.
sort	Sorting a whole file or a group of records into a new order, usually prior to printing them out.
calc	Doing calculations on screen
output	Producing printed output in the form of lists or as one record per page; controlling the appearance of the output; showing lists on the screen; making lists on disk for use with the word processor.

If your work includes some of these activities (and if it doesn't then perhaps you don't need Superbase at all) then thinking about them carefully before you begin to design your system is bound to produce greater efficiency all round.

File Design

If I had to pick the most important discussion in the book, this could be it. If you don't think clearly about the raw material of a system -- the data -- you'll never manage to get the best out of it. Imagine the chaos if that ubiquitous example of a file, the telephone book, was printed in the order of the numbers, or contained no address information. But if some Orwellian police force wanted to get a name from a number, how useful a directory in number order would be. The key to the business of file design lies in the question "What do I want to do with the file?" but since there are as many individual purposes as there are individuals, I shall have to offer only general suggestions to help you find your own answer to the problem.

Your Existing Files

The first step in designing a file is to identify the general area of the application. You can make a start on this by considering the manual files you keep already. (You should be clear about the meanings of the words "file" and "records"; the former is a collection of the latter. A good image is of a card index box; the box is the "file", and each card is a "record".) Do you have a name and address file for mailings only? Or is it in fact a customer file that will be holding account balances, which are worked out from a separate file of invoices? Or perhaps you only keep the name and address on each invoice, and you will devise a way of adding up the balance for each customer when you print out a list. So, make a rough analysis of the different categories into

which your data falls; there will be time to refine them later if necessary. Here is an example of a typical business database comprising five files:

```
Database: WORK

customers
invoices
inventory
suppliers
assets
```

A home database might contain the following files:

```
Database: HOME

records
books
bills
inventory
addresses
```

Working out the main areas is usually not too hard. The next step is to figure out exactly what goes into each file -- not so easy.

Working Backwards from Printout

Think about the lists or other printed output, such as personalized standard letters, you want to produce. There is always some kind of internal order in the output, and this can be used to help determine the components of the file that is to produce the output. First of all, identify the basic unit of the output. This could be a name and address, or the details of one of the products a business sells, or any one of thousands of logical groupings. Then break the basic unit down into the individual elements. This is sometimes not as straightforward as it seems.

As an example, let's consider a file of names and addresses that is to be used to generate labels and data for use in a mail merge. A name and address may seem to be nothing more than the collection of these items, or, as we generally refer to them, "fields":

```
name
street
city
state
zip
```

(I hope British readers will forgive my adoption of the American convention for classifying addresses.)

A closer look reveals that the first element, name, is in fact an assemblage of four separate fields:

```
title (Mr/Ms etc.)
initials
surname
suffix (MD/PhD/Jr etc.)
```

This illustrates a very important part of the process of file design: breaking the data down into its smallest components. Doing this gives you maximum flexibility, a set of basic bricks to manipulate rather than awkward combinations that limit what you can do. For example, the name and address format we've seen so far is all right for a label. But suppose you want to address the recipient personally (which is the whole idea behind direct mail). You don't want to write "Dear R. B. Jones" but the more personal "Dear Bob". To do this you need another field:

```
first name
```

So now the record format contains all these fields:

```
title (Mr/Ms etc.)
initials
surname
first name
suffix (MD/PhD/Jr etc.) street
city
state
zip
```

This may be all the information that is needed to produce the desired result of a file of data for use in a mail merge. But that is by no means the end of the process of designing this file. So far we have looked only at what the record must contain to produce the desired result of a flexible mail merge. We must also consider whether there may be a need for other fields that will increase the power of the system. This is hard to do unless you know more about Superbase's capabilities. The two main functions that no manual system can duplicate are the 'find' and 'sort' menu options. I'll be looking at both 'find' and 'sort' again later on, but it's worth considering them both here briefly in the context of file design.

Fields that Help You Search the File

The first, 'find', allows you to carry out automatic searches of the database, specifying a large number of values, or "criteria" as these are called in the manual, which determine whether a record is to be selected or not. In order for 'find' to work effectively, you must arrange for fields that can be used for meaningful searches to appear in the record. In the name and address file, you could ask Superbase to 'find' all the individuals in a certain city, by entering the name of the city. Similarly with searches by state, or even searches using the first three characters of the zip code. But more is possible.

The crucial realization is that you don't need to restrict the

fields in the record to the ones that will be printed. For example, you may know about the position of the individual -- vice-president or managing director or mayor. If you include this information you can then search the file for all the records that match the criterion "mayor", and send letters only to mayors. Or, you may be dealing with a list of customers. It will help to know such things as when you last wrote to a customer, and what category of customer each one is: good, average or poor, for instance, which you might represent by a one letter code, A, B or C. So you add three fields to the format: 'position', 'category' and 'date last mailed'. Immediately you have this information on file, you can discriminate among the records and 'find' only the ones you want to process, such as:

all managing directors
who live in the London area
are rated as category A
and have not received a letter for two months

You need to think ahead. Even if you don't know the information that could be useful for a 'find' now, maybe you will in a few months. In any case, you can always add new fields to the end of the record if need be, or in extreme cases reorganize the structure of the file completely.

Fields that Help You Change the Order of the Printout

Another exceptionally useful function is the 'sort' option. Superbase stores records in the alphabetic order of the index key field. This is fine as long as the final order of a printed list is the same. But quite frequently this is not sufficient, even though alphabetic order may be a necessary component of the desired result. In the above example of names and addresses, the records are most conveniently stored in the order of the person's surname, so that you can call up an individual record easily to look at it or change its details. But it is quite likely that mailings will be done on the basis of the zip code or post code: this allows you to send out mail on a regional basis, and you may earn a discount from the post office if you give the letters to them already sorted.

Superbase lets you change the order in which you output the records; this is called sorting. (Actually, the database is not affected by the 'sort' operation; as I explain in a later section, 'sort' uses an efficient and economical system of temporary indexing to achieve its ends.) As with the 'find' command I discussed above, 'sort' can use any field in the record (in fact, many fields may be used at the same time) to change the order of the records. Once you realize this you may wish to include some fields especially for 'sort' to use. For example, if your name and address file becomes a file of prospective customers, you may want to classify them by a field called 'status', which allows you to sort the records into status groups, and perhaps print them out that way.

Summary

File design is crucial to the success of your Superbase application. To start with, you must identify the general data groupings you will need: your files. Each file contains a set of individual records of an identical format, each one holding different data. The data is kept in separate fields, each of which must have a clear function. When deciding on the fields that will constitute a file format, follow some simple rules:

1. Break the record down into the smallest components of data. This gives maximum flexibility.

2. Work backwards from the items you want to see on your printouts or screen displays.

3. Consider whether you want to be able to process groups of records from within the file. If you do, will you be able to 'find' them using the existing fields, or will you need to add some fields specially for classification and database searching?

4. Will you be able to obtain the desired order of printout from the existing fields? If not, you may have to add a special field -- or will one of the classification fields do the job?

You should end up with a first draft of the list of fields for the format of each file. Our example looks like this:

```
title (Mr/Ms etc.)
initials
surname
first name
suffix (MD/PhD/Jr etc.)
street
city
state
zip
position
category
date last mailed
status
```

You may well need to modify this list as you learn more about the details of the different types of fields that can be used in a format. Most important of these is the index key field, which I mentioned briefly above, and which leads us into a closer look at the process of defining the format.

Ordering the File: the Index Key Field

The internal order of the file, comparable to the order of cards in a card index box, is determined by the index key field, commonly called the key field or just the key. Each record has one

key field. The key is stored both separately and in the record itself, a relationship like that of an entry in a book index to the page on which the reference occurs. The key is just a way for the system to look up a record quickly. Although Superbase allows you to have several records all with the same index key entry (such as the surname "Smith"), we do not recommend this way of using the key field. The subject of "duplicate keys", as identical index key entries are called, is discussed both in the Superbase Manual and elsewhere in this book, and I shall not go into it further here. In this book, unless otherwise indicated, all index key fields are assumed to be for unique entries only.

The index key field is the crux of operating efficiency. This is no less true in simple systems than in complex programmed multi-file applications. When you add a record to a file, Superbase makes an entry in the index. Later, when you go to output records from the file, the order of entries as they occur in the index determines the order in which the records appear. So, you add records for Williams, Baker, Young, and Novak, in that order, and they will be output in the order:

```
Baker
Novak
Williams
Young
```

General Rules for Key Formation

Key Field Length

Set the field length for the key as short as possible. This reduces the disk space used for storing the index and for compiling lists of keys with 'find' and 'sort'.

Allowed Characters and Character Sequence

Avoid using characters such as dash (-) or slash (/). Stick to the characters of the alphabet and numbers. The order of the main character groups is:

```
space
numbers 0 - 9
capital letters
lower case letters
```

These priorities are illustrated by this list of index keys, which would be stored in the order:

```
1smith
110smith
2 smith
2smith
2 Smith.a
2 smith.b
```

```
Smith.a
smith.b
```

Unless the '.a' and '.b' suffixes are used, Superbase will interpret the keys in each of the last two pairs as duplicates. If duplicates are being used, 'Smith' will always precede 'smith'.

Dates as Keys

You can't use Superbase's special date fields as keys. To use a date as a key, first recast it into a form that follows the rules of sequence. For example, 23rd November 1985 becomes '851123'. All such keys must be the same length, so dates prior to the 10th of the month and months prior to October must be padded with a zero: July 4, 1986 becomes '860704'.

Numeric Keys

If you want to use numbers as index keys, you must ensure that every key is the same length. If you don't, the normal cardinal number sequence:

```
1
2
12
24
154
```

will be stored as:

```
1
12
154
2
24
```

Superbase compares each character position in turn, ignoring the actual value of the number. To make keys an even length, pad them with leading zeroes:

```
001
002
012
024
154
```

This would allow up to 999 keys; if you need more, increase the number of zeroes you allow -- but figure this out in advance as it is quite a job to change the keys of a large number of existing records.

Using Suffixes to Make Keys Unique

One of the large issues is how to handle keys where the records are to be stored in order of people's surnames. Do not go for duplicate keys -- this creates more problems than it solves. The best solution is to make the name an ordinary text field, with the separate index key being constructed from a combination of the last name and a unique numeric suffix. For example, the records for four people called 'smith' would have the keys:

```
smith01
smith02
smith03
smith04
```

Leading zeroes are used to ensure that all the suffixes will be the same length, allowing for up 99 smiths in the file. Suffixes do not have to be numeric -- a two character alphabetic code in the sequence from 'aa' to 'zz' would be just as effective, but you might want to use a dot to separate the suffix from the rest of the name:

```
smi.aa
smi.ab
smi.ac
```

In this example I have deliberately not used the whole name. Usually no more than six characters are needed to differentiate one name from another, and you can afford to abbreviate it. However, beware of the pitfalls: 'williams' and 'williamson' can only be differentiated at the ninth character.

A commonly raised difficulty is the problem of knowing where you are in the sequence if you have to add many keys based on the same name, but not at the same time. Experience suggests that the best way is to use Superbase's own logic to get quickly to the end of the sequence, and then write down the new code before entering the record. I'll go through the actual steps in Chapter 4.

Keys Need Not Be Obviously Meaningful

Although in many cases you will want to ensure that the index key is meaningful, so that if Mr. Gregory telephones to complain you can call his record onto the screen merely by typing in 'gregory', it may be more useful to have meaningless keys. Take a file of parts for an auto business. The description of any one part -- the nearest you can get to its "name" -- may be exactly the same as for hundreds of other parts. It makes more sense to use the part number, which is by definition unique. This also allows you to print out a list of the file by part number without having to sort it first, as the key order is the required order. The drawback is that you have to know the part number to call the record back, and so you will need to have a list printed in description order as well so that you can look up the part number.

Structured Keys

Keys can be much less meaningful to a casual observer than a part number. You may prefer to devise a coded scheme of your own to help you keep records in an order that cuts down the need for the 'find' and 'sort' functions. Take a key such as 'd.citi.851101.1423'. Here a banker who regularly makes deposits or arranges loans with a variety of banks has adopted a scheme based on transaction type, four letter bank code, the date, and a unique suffix derived from the 24 hour clock in case more than one transaction of the same type to the same bank occurs on the same date.

d — Transaction type: d for deposit, l for loan. A printout of the file will list all the deposits and then all the loans.

citi — Bank code. All deposits for Citibank will be grouped together, as will all loans.

851101 — The date. Recast into numeric sequence, it ensures that the printout lists all transactions of the same type in date order for each bank.

1423 — The time. No two transactions to the same bank can occur at the same time, so this is a useful way of ensuring uniqueness as well as preserving the actual order of events in the printout.

Structured keys really come into their own with programmed applications, but even a little experience of 'find' will show you how helpful they can be. Of course, any element of data can be used as part of such a key: codes for multiple transaction types, gender identifiers, active/inactive flags, cross-references to other files in a multi-file system. At this stage, all you need to do is be sure you understand the principle, so that even if your first index keys are simple, you will be able to incorporate a structured element if it's obviously appropriate to your data.

Before going on to consider the other types of field allowed in Superbase records, come to a provisional decision about what kind of index keys you want. Don't be too absolute, as the next chapter may suggest changes to your first ideas.

CHAPTER 2
DESIGNING THE FIELDS

Text Fields

The text field is the workhorse of the system. If you're not sure which type to make a field, go for text. Almost any characters can be stored in a text field, except for the illegal double quotes (inverted commas). However, if you use the following, you may invalidate some of the ways in which 'find' represents its search criteria:

? * / - > <

Notice the dash in particular; this tends to be overlooked when it appears as a hyphen in a name.

Text fields can be up to 255 characters long, thus extending across three 80 column lines. Some users complain when they find that Superbase does not use wordwrap! We tell them gently that it's not a word processor, and suggest that if they need one they buy Superscript. Seriously though, it's very tricky to format the contents of text fields in a printout, so if that's a requirement you should consider a different approach. A compromise solution is to set up a number of text fields and be careful when you type the data in. Here's an example that was needed when the user had to print out a list of names stored in text fields, but did not want to have one field per name:

The output:

Tony Rome	Peter Thomas
Bernard Small	Daniel Grossmund
Liz Schwer	Cecil Watson

The text fields:

```
f1 <Tony Rome                    Peter Thomas    >
f2 <Bernard Small                Daniel Grossmund>
f3 <Liz Schwer                   Cecil Watson    >
```

But notice that if Tony Rome was deleted, Peter Thomas would appear in his place, as Superbase strips away any leading spaces. Also, if Daniel Grossmund was eliminated, there is no way to close up the gap by moving up data from the next field. You would have to retype it.

Superbase also strips away any spaces between the last character of data and the end of the text field. So a field that looked like:

```
name <Bo                                          >
```

would be stored as just two characters. This is called variable length storage, and allows Superbase to make good use of the small disk capacities it often has to work with. So you can afford to be

generous when setting up text fields. Always allow a few extra spaces for that longer exception to the rule. (Even if you get it wrong, it's easy to fix: re-enter the 'format' option, position the cursor inside the field, insert some spaces into it, then finish the function normally.) However, if you become too generous, you will find that when you come to finish the formatting function Superbase gives you the "Record Too Long" error message. If this happens there's nothing for it but to reduce the length of some of the fields.

One disadvantage of long text fields only appears when you print them out (or view them with 'output'). Superbase prints the whole length of the field, even if it's empty or has only one or two characters in it, unless you include an instruction to shorten it. Keeping the field length short to begin with can make this extra instruction unnecessary.

In general, use text fields for any category or analysis fields you're going to need for 'find'. Make them longer than the length of the category code itself, as you want room to type alternatives in when specifying the 'find' criteria:

```
    code    <a/b>
```

The field needs to be three characters long even though it will only be used to store a single letter.

A feature of Superbase is the ability to store numbers in text fields and refer to them in arithmetic processing. I suggest you avoid this except in the simplest applications, as it reduces flexibility. Clearly, this feature conflicts with the principle of breaking the data down into its smallest components.

Numeric Fields

Numeric fields are relatively simple creatures -- all they do is store numbers, right aligned, to the decimal point if you specify one.

Numeric fields are automatically checked when you enter data into them, to ensure that no non-numeric data creeps in. This is very useful, as you can be sure when using 'find' or calculating a total that the results will not be invalidated by a bad piece of data. There is no provision for checking the range of input data (for example, whether a number is between one and ten), without programming the input operation, as explained in Chapter 9.

Date Fields

The properties of date fields are also few. A date must be within the 20th century, and must be entered in one of two forms:

```
        ddmmmyy
  or    mmmddyy
```

The form is not determined when you set up the date field. Normally a date field is seven characters long, and displays the date in one of the forms shown above. If you wish, you can extend the length of the field to show the day of the week:

1. Set a normal date field, pressing return at the end of it.
2. Cursor left into the field.
3. Insert four spaces.
4. Leave the field.

Dates are stored internally in julian form, which means they are basically treated as numbers, not characters. This imposes some limitations on how you can process date fields.

Like numeric fields, date fields are checked on input. You cannot enter an invalid date. If you have an eleven character field, the word "bad" will appear in place of the day of the week if you try to input an invalid date. As with numeric fields, range checking beyond the simple question of validity can only be achieved through programming.

If you need to process dates in a way that conflicts with Superbase's rules, you must use another kind of field. Usually a text field is sufficient for storing dates outside the 20th century, such as 5th November, 1605, or even for 20th century dates where the form has to include the full year, such as "1986". Such a field will be no good for sorting, though, and if you need to sort on non-20th century dates you should recast them into numeric form and store them as in "16051105" in either a numeric or a text field; this could appear right next to the text form of the date.

Constant Fields

This type of field can be very useful and is underused. Basically a constant field is a text field with a preset value that is presented each time you add a new record to the file. Typically, you use constant fields to hold data such as the current tax rate, or a description that occurs in more than 50% of the records in the file. When you add a record, the cursor moves into the constant field, which is already showing the preset value, giving you the option to accept it by pressing return.

However, you don't have to accept the presented value. You can edit it. This allows you to set up a constant field to hold the most frequent content of a field, changing it only when necessary. Suppose a file of names and addresses referred predominantly to women. You could set the 'title' field to "Ms", and you would only need to alter it whenever a man's record was added. Where the content is longer, considerable typing time can be saved. A further refinement has the constant field holding partial data which is always the same, but is completed with different data for each record. A cautionary word -- the maximum length of a constant field is 32 characters.

You can change the semi-permanent content of a constant field. If you do this (by editing the format), all future records will hold the new value; old records will not be affected. So, if the tax rate changes, your old records will still hold the old rate.

Result Fields

Result fields are one of the most powerful features of the Superbase system. Properly used, they can perform a number of important tasks and significantly reduce effort in other areas, such as reporting and analysing data. Many people find it helpful to understand a Superbase result field as a kind of formula, comparable to the cell based formula of a spreadsheet. Rather confusingly, Superbase calls the formula a calculation, but I think that for the purposes of this discussion formula is better. A result field formula can use many of the same components as a spreadsheet formula:

arithmetic functions
arithmetic operators
references to other data elements
numeric values
external variable values

Arithmetic Functions

These are neither as extensive nor as specific as those in the average spreadsheet. There are no 'sum' or 'NPV' functions, for instance. The available functions are those of the BASIC programming language:

abs	log
asc	sgn
atn	sin
cos	sqr
exp	tan
int	val
len	

BASIC string functions are not allowed. In practice, few of these functions are used, certainly not in the majority of business or home applications. Nevertheless, they have their uses, as some of the examples later on in the book will demonstrate.

Arithmetic Operators

The allowed operators are also those of the BASIC language:

+	>
-	=
/	(
*	)
	<

The same evaluation hierarchy and the same rules of permissible combinations apply here as do in BASIC, and the introduction to your BASIC programming manual will give all the necessary information.

References to Other Data Elements

Other data elements means other fields in the record. You cannot refer to the result field itself within a formula, so if the formula for the field [total] was set up like this, while it would not be illegal, it would produce a loop within the field:

```
[total]:  [total]+[amount]
```

(In these examples, I place the name of the result field on the left, and the formula for it on the right. Actual definition of the formulas is part of the 'format' function.) Such an expression can form part of a 'batch' updating command, and I shall explain later how to use 'batch' to maintain running totals in records.

Usually a result field formula performs a simple calculation such as adding two fields together:

```
[total]:  [amount1]+[amount2]
```

This can be extended to include many common calculations:

```
[total]:  ([amount]*[quantity])-[discount]
```

Note that as date fields hold the date in julian or numeric form, they can be used in result fields to compute the number of days between two dates.

```
[interval]:  [date2]-[date1]
```

The number of days obtained in such a result field could be used in another result to compute interest.

As this implies, result fields can refer to other result fields:

```
[total]:  [subtotal1]+[subtotal2]+[subtotal3]+[subtotal4]
```

You must be careful not to allow division by zero, which may be hard to trap even if you have a version of Superbase that allows error trapping (C128 or Apple). This error can occur in any Superbase version when a result field is used to compute an average figure:

```
[average]:  ([f1]+[f2]+[f3])/[count]
```

If [count] is zero the error will occur. The only way round this is to use a special formula, which is quite complex, but too useful to be omitted:

```
[average]:  [total]/(([count]=0)+[count])-([count]=0)
            *[total]
```

Here, [total] is the sum of the fields to be divided, and [count] is the number of occurrences. In BASIC, an expression of the form (a=b) actually has the value -1 or 0, depending on whether it is true or false. So a BASIC program of this form will print either -1 or 0 depending on the value of a input:

```
10 input a
20 print (a>10)
30 goto 10
```

Notice that here I use the '>' operator -- all operators are valid bases for the test. We can use such a test in a Superbase result formula. In the one above, ([count]=0) evaluates to -1 if [count] has the value of zero, or 0 if [count] has any other value. If [count] is zero, the expression ensures that the divisor cannot be zero; but if [count] is not zero, the expression reduces itself to allow the number in [count] to be used as divisor.

A common fault when entering a formula is the "Formula Too Complex" error. This usually means that there are too many sets of parentheses in an order that BASIC cannot evaluate accurately. You must break your formula down into subformulas, if necessary treating them as "working fields" on the record format, and combine the intermediate results to get your final result. I give some examples below.

Numeric Values

Often, the result field formula is most efficiently constructed with the aid of a fixed numeric element. An example is the calculation of a percentage increase of 12 percent:

```
[total]:  [amount]*1.12
```

A decrease of 10 percent would be rendered:

```
[total]:  [amount]*.9
```

But what if the percentage is held in a numeric field or a constant field in the record? A slightly more elaborate formula produces the desired result dynamically:

```
[total]:  [amount]*(1+([percent]/100))
```

This also handles decreases if they are input in unary negative form, e.g. -12.

If you want to obtain a percentage of a field, this is the formula you need:

```
[percentage]:  [amount]*([percent]/100)
```

Another important use of numeric elements is to produce a conditional result depending on the value input into a field. This is achieved through a combination of work fields and Boolean tests as used above in the division by zero formula.

The following example produces a labour charge based on an hourly rate and a number of hours and minutes worked, the latter rounded up to the nearest quarter of an hour. The rounding up is achieved through a series of four result fields which produce 25%, 50%, 75% or 100% of an hour's charge depending on which quarter of the hour the number of minutes falls into:

```
[hours]                    These fields accept
[minutes]                  the input data.
[rate]

[quarter1]:  (0.25*([minutes]>0)*([minutes]<16))*[rate]
[quarter2]:  (0.5*([minutes]>15)*([minutes]<31))*[rate]
[quarter3]:  (0.75*([minutes]>30)*([minutes]<46))*[rate]
[quarter4]:  (1*([minutes]>45)*([minutes]<61))*[rate]


[charge]:    ([hours]*[rate])+[quarter1]+[quarter2]+
             [quarter3]+[quarter4]
```

Exactly the same principle would apply to the construction of a discount table within a record, where the discount percentage would depend on which field produced a non-zero result:

```
[quantity]
[price]

[break1]: (0.4*abs([quantity]>500))*[price]
[break2]: (0.5*([quantity]>400)*([quantity]<501))*[price]
[break3]: (0.6*([quantity]>300)*([quantity]<401))*[price]
[break4]: (0.7*([quantity]>200)*([quantity]<301))*[price]
[break5]: (0.8*([quantity]>100)*([quantity]<201))*[price]
[break6]: ([quantity]>0)*([quantity]<101)*[price]

[disc-price]: [break1]+[break2]+[break3]+[break4]
              +[break5]+[break6]

[order-value]: [quantity]*[disc-price]
```

Here, if [quantity] is greater than 500, [break1] produces a figure which is 40% of [price]. Since all the other [break] fields will then produce zero, the new price, equivalent to a sixty percent discount, is obtained in [disc-price] by adding all the [break] fields. When one [break] field produces a positive result, the others will always produce zero, so only the contents of one break field are used to produce the final result.

To complicate things still further, you could set up the table inside the record itself, with constant fields to hold the percentages of the original price that apply to the break point

values, as well as the break point values themselves:

```
p1 < .4> p2 < .5> p3 < .6> p4 < .7> p5 < .8> p6 <1.0>
t1 <500> t2 <400> t3 <300> t4 <200> t5 <100> t6 <  0>
```

The field [break2] would then look like this:

```
[break2]: ([p2]*([quantity]>[t2])*([quantity]<[t1]+1))
                                                *[price]
```

The discount table is wholly dynamic, and you need only make changes to the table fields to alter the automatic discount calculations.

External Variable Values

This concept only applies to programmed systems, but since it follows on so logically from the last topic I have decided to deal with it here. Superbase allows you to set up variables in memory. These can then be referred to from within a formula. However, if the values are not set prior to entering or editing record data, they will be taken as zero. For example, if a program set up two tables of values to correspond to the [t] and [p] fields above, they might look like this

```
10 p(1)=.4:p(2)=.5:p(3)=.6:p(4)=.7:p(5)=.8
20 t(1)=500:t(2)=400:t(3)=300:t(4)=200:t(5)=100
```

The [break2] field would look like this:

```
[break2]: (p(2)*([quantity]>t(2))*([quantity]<t(1)+1))
                                                *[price]
```

PROGRAMMER'S CHALLENGE: *Set up a result field to generate a subscript from a number input to a record field, and use it to reference an external array, in such a way that only one result field is needed to determine the discounted price.*

Positioning Result Fields

When several intermediate result fields are being used to determine a final result, the ordering of the fields on the screen can affect the internal evaluation. If the intermediate fields occur after the result, no value will appear in the result until the cursor has moved past them. If such result fields are the final fields on a screen, the system will jump to the next screen (if there is one) before you have a chance to see the final result.

Changing the Format

Whenever you set up a record format with result fields, print out the 'status' of the file. This is best achieved with the command

line 'print:status:display' (you may need 'print:maintain s:display' on earlier versions of the program). I advise this because if you return to the format to insert or delete fields, which is not uncommon when first setting up a system, changes to the format can scramble a carefully designed structure. For example, if you have three result fields:

```
[result1]
[result2]
[result3]
```

and you delete [result1], when you come to finish the format you will find that the sequence of the formulas will have been disrupted. In such a case, reference to a printout is essential.

Outputting Formulas

This is rather a misnomer, but it is important to realize that Superbase allows you to do things like multiply and add fields as you print them out. If your requirement is only to see the results of calculations on paper, you may wish to avoid putting result fields into the record format, and instead incorporate them into an 'output' command line or program. This can save disk space for storing the result, and also reduces the size of the file format, which allows slightly faster operation.

Calendar Fields (Period Fields)

Much of what was said about result fields is also true for calendar fields, which are simply result fields for dates. There is no requirement for complex arithmetic formulas where dates are concerned. Calendar fields always express the result as a date, not as a number. The most common application is the generation of one date from another:

```
[next-appt]: [today]+90
```

This finds the date 90 days from the date input into the [today] field. Of course the figure 90 could be replaced by the value in another field:

```
[next-appt]: [today]+[interval]
```

If you program the system, you could set the value of today's date into a numeric variable 'day', and use this formula to enter it into the record automatically:

```
[today]: day
```

Whenever the record is stored, the current value of 'day' will be replaced in the record, so you would have to be careful when editing records at a later date.

Forced Fields

The forced field is another underused feature of Superbase. It has its uses, for example the fact that the key field of a record is automatically made into a forced field. This ensures that the record cannot be stored in the file unless it has a key -- an obvious requirement. If you think of the forced field option as a way of validating your data, its usefulness becomes clear. Any field can be set as forced, and this will guarantee that the operator cannot leave it empty. One thing you cannot do with a forced field is attach a range check to it. You can do range checks in Superbase, but they must be programmed in.

Replica Fields

A replica field is one that has the same name as another field. The original field always precedes the replica or replicas in its position in the format. The commonest use for replicas is to carry forward a reference such as the index key field across a multi-screen format, so that the operator can always see it. A replica field uses only one byte of storage, but it does count as one of the 127 fields you are allowed per record. You cannot edit replica fields, nor can you use them as result fields or index key fields.

CHAPTER 3
MORE DESIGN CONSIDERATIONS

Field Names

Unless you want replica fields, you must ensure that all the field names within a record format are unique. There are some important rules to be followed.

Identifying the Actual Field Name

Superbase takes the first group of characters to the immediate left of the start of the field as the field name. The limit of the group is set by the first space encountered going from right to left. For example, here the field name is [name]:

```
Middle or family name <                    >
```

But here it is [family-name]:

```
Middle or family-name <                    >
```

The maximum length of a field name is 12 characters. So if you set up a field like this:

```
Middle-or-family-name <                    >
```

Superbase will take the field name to be [middle-or-fa]. It takes the first 12 characters to the right of the first space encountered, which in this case occurs before the "m" of "middle".

Legal Characters

We strongly advise you to use only letters, numbers, and the hyphen and period (full stop) when naming fields. Some other characters are acceptable, but you must never use the double quotation mark -- " -- and these should be avoided:

```
[ ] ( ) / * + - =
```

Likewise, you can use any word in a field name. Even BASIC or Superbase reserved words may be used, as the method of referring to fields ensures that the system can always tell the difference.

Referring to Fields

Whenever you refer to a field, you must place its name between square brackets. Thus, 'name' on the screen becomes [name] in a command line or program or result field formula. If you don't do this, Superbase will treat the name as a BASIC numeric variable, and output or process it with the value zero.

Naming Conventions

There aren't any. You can call your fields whatever you want. However, I recommend strongly that you give fields meaningful names wherever possible. When you are trying to remember the fields as you type the parameters of an 'output' command line, it helps if they are not cryptic gibberish. (On the other hand, some people maintain that the more bizarre the name the easier it is to remember. Do your own thing.)

One convention I always use myself is a standard name for the key field. I almost always call it [code], although I combine [code] with the appropriate description according to the purpose of the file. Here are some examples of the code as it might appear in five different record formats:

```
Customer code  <          >

Product code   <          >

Transaction code  <       >

Account code   <          >

Supplier code  <          >
```

Very Short Names

There are two situations when field names should be kept as short as possible. The first is when you have to place many names in a row across the screen. This may be necessary to preserve the intelligibility or logic of the input screen -- a very important consideration. However, the same consideration demands that you provide the operator with some means of understanding the purpose of the field. One method is to provide column headings:

```
Quantity Description          Price      Tax       Total
-------------------------------------------------------------------
q1<    > d1<                  > p1<      > v1<     > t1<        >
q2<    > d2<                  > p2<      > v2<     > t2<        >
q3<    > d3<                  > p3<      > v3<     > t3<        >
q4<    > d4<                  > p4<      > v4<     > t4<        >
q5<    > d5<                  > p5<      > v5<     > t5<        >
q6<    > d6<                  > p6<      > v6<     > t6<        >
```

It is easy to see that this format is that of an invoice. Even on a 40 column screen, this technique is possible without too much loss of clarity:

```
Quantity Description
Price      Tax       Total
--------------------------------
q1<    > d1<                    >
p1<      > v1<     > t1<       >
--------------------------------
q2<    > d2<                    >
p2<      > v2<     > t2<       >
--------------------------------
q3<    > d3<                    >
p3<      > v3<     > t3<       >
--------------------------------
q4<    > d4<                    >
p4<      > v4<     > t4<       >
--------------------------------
q5<    > d5<                    >
p5<      > v5<     > t5<       >
--------------------------------
q6<    > d6<                    >
p6<      > v6<     > t6<       >
```

Very short field names are useful if you want to 'output' a large number of fields with one command line. Above, we have six invoice lines, each of five fields, a total of 30. When you output, two square brackets must be added to the field name, so even though I have limited each name to two characters, 120 characters are needed to refer to all the fields. This is more than the two line command area of a 40 column screen can handle, though well within the capability of an 80 column screen. To achieve the desired result, you would have to program the system and extend the length of the command line with the 'plus' instruction. This is one of the easier programming techniques, which I explain in Chapter 7.

Field Names in Programs

Also dealt with later in the programming sections is the use of field names in program loops. In order to avoid explicitly naming every field when an identical operation is being performed on all the fields, you must combine an incrementing numeric variable such as the control variable in a 'for ... next' loop with the name of the field. The field names themselves must obviously be completely regular so that the program can predict the form of the name correctly.

Finally, if you have a number of identical or similar files in your database or databases, consider making the field names the same in each. If you do this carefully, you will be able to develop and store command lines and programs which can work with a multitude of files, saving much effort and disk space.

Visual Design

Even though you may be the only person who uses your database, you will come to appreciate the benefits of designing the visual appearance of your record formats with some care. Sooner or later you will find yourself looking at a screen that you have not used for some time. What will you do if you can't remember what that field called "tx1" was supposed to be for? If others are to use the files you set up, the need for effective communication in this area is paramount.

In general, follow the principles that everything should be as self-explanatory as possible, and as easy to read as possible. To this end, make your screens as regular as possible. Here is an example of an irregular screen:

```
                                                  date    <       >
         name    <                                        >
CUTOMER code  <          >
 ADDRESS   <                                             >
 city      <                    >
 cty      <                                            >
             Pc   <           >     tel.          <        >
      categ. a <  > categ. b <  >
```

This is breaking all the rules -- and I have seen worse. The fields begin at seemingly random column positions, there is no consistency in capitalization, and some of the abbreviations are idiosyncratic. There are no divisions corresponding to function. No use is made of the 1000 character allowance for purely descriptive text, which covers everything except field names.

Careless spelling completes the mess. Yet it is so easy to produce a clean design:

```
CUSTOMER ADDRESSES                                        Screen 1
-------------------------------------------------------------------
Customer Code   <          >                  Date  <       >
-------------------------------------------------------------------
Name      <                              >

Address   <                              >

City      <                              >

County    <                              >

Postcode  <                >

Telephone <                >
-------------------------------------------------------------------
                    Analysis Categories

Status <    >                            Activity <    >
                                                            +...
```

Here are some guidelines that should help you arrive at satisfactory layouts:

1. Headings. Treat the screen as a form, and give it a clear title, separated from the part of the screen used for data entry. Don't use the word "file"; it's redundant. If your format has more than one screen, indicate the fact with a screen number, so you will always know where you are. This is essential for complex formats with dozens of fields. Multi-screen formats should have the index key reproduced on each screen, together with any other data necessary to identify the record. Use replica fields to duplicate data.

2. Separators. The aim is to improve legibility. Use graphic characters if they are available, or dashes or equals signs if you prefer. Graphics characters allow you to design enclosed boxes for different groups of fields. Blank lines are also effective, so use them to avoid a crowded feel. You have up to 1000 characters to play with, though these don't go as far as you might think.

3. Highlights and use of colour. Reverse video is useful to draw attention to a particular field or area of the screen, but don't overdo it or the effect will be lost. If you have a colour monitor, you should definitely set up each file with its own colour scheme for background, lettering, and border if available. Then when you select the file or return to the screen you can tell without reading which one it is. Ensure

that the lettering stands out clearly. If you like you can colour code your files to suit the application: red for debtors, black for creditors.

3. Field groupings. Here the aim is intelligibility. Often a record stores several types of information, for example system data such as the index key, sorting data such as dates or transaction types, static data such as a name and address, dynamic data such as an account balance, analysis data such as search criteria, and textual data such as comments. Each type of data should be placed separately on the screen, or if necessary on a separate screen.

4. Multiple screens. Don't hesitate to use them. To obtain the next or previous screen while you're formatting, the commands are 'f1 +' or 'f1 -' respectively; when viewing with 'select' use just '+' or '-' to switch screens. In the example above, I use '+...' to show that there is another screen and remind the user how to obtain it. The '-...' at the lower left of the second screen would prompt for return to the previous one. It often makes sense to keep key data on the first screen, and use other screens for detail or comment fields, especially when you may want a degree of confidentiality for comments like "Jones is a lousy credit risk". For very long records that involve complex result field calculations, you may want to keep a summary of the final results on the first screen, using replica fields.

5. Field names. These have been discussed above, but I want to emphasize the technique of providing subheadings to improve intelligibility. In the example I used "Analysis Categories" as a subheading. You can combine subheadings with column headings to even better effect, especially where the field names have to be abbreviated:

```
------------------- Mailing Information Record -----------------
Date           Package          Reply      Follow-up  Order Value
-----------------------------------------------------------------
d1 <         > p1 <           > r1 <     > f1 <      > v1 <       >
d2 <         > p2 <           > r2 <     > f2 <      > v2 <       >
d3 <         > p3 <           > r3 <     > f3 <      > v3 <       >
d4 <         > p4 <           > r4 <     > f4 <      > v4 <       >
d5 <         > p5 <           > r5 <     > f5 <      > v5 <       >
d6 <         > p6 <           > r6 <     > f6 <      > v6 <       >
d7 <         > p7 <           > r7 <     > f7 <      > v7 <       >
```

6. Prompts. Even if you will not be using a program to prompt the operator with the correct input, you can still include key information on the format itself. Suppose a key field consists of a numeric date and a letter code to make it unique. You can indicate the required form of input on the screen like this:

```
Code  <          >   Use date format yymmdd plus 2 letter
                     alpha code, e.g. 851121aa
```

There are one or two drawbacks to using the formatting facilities extensively. First, the file format takes longer to load when you select the file. Second, Superbase allows up to three file formats to be held in memory simultaneously, so that changing from one to the other is virtually instantaneous. However, the 1000 characters for descriptive text has to be shared out between all the files in memory, and if one of them hogs most of the descriptive text then Superbase has to reload any of the other formats when it's selected. This can slow things down, especially in a programmed application that uses several files; it doesn't matter so much if you use one file at a time for fairly long periods.

PROGRAMMER'S TIP. *Combine standard formats with your own displays generated from inside the program. This allows you to use flashing effects and temporary windows for error messages, prompts, file look-ups, etc., while keeping the actual formats small enough for three of them to be held in memory. When displaying error messages ensure that every message appears in the same screen location, perhaps near the top or bottom of the screen.*

The Database Structure

While I am not proposing to impart a knowledge of the internal workings of Superbase, I know from experience that a clear understanding of the most important components and the relationships between them will help many people to avoid a number of common errors. The definitions below provide a hierarchy of Superbase entities from database down to field level.

Database — The mass of data from all the files that appear on the database catalog, which the 'file' command displays. On the disk, the database name appears in upper case, and occupies only one block. This is because the database name contains only pointer information, telling the system where each of the files begins.

File — The collection of records identified with a common format and name, such as "addresses" or "invoices". A single entry on the database catalog.

File format — The same as "record format" and "file definition". The different terms are used to indicate the point of view. "Record format" relates the design of a data input screen to the user who thinks in terms of individual records. "File format" generalizes this term and applies it to the collection of records known as a file, still retaining the sense of a designed layout.

File definition — The file format understood as a separate entity from the database. All the data that you put onto the screen when you design a record format, including field names, positions, types, result field formulas and descriptive text, is stored on the disk as a file of data. It is essentially the same as other disk files such as word processed documents or programs, except that only Superbase can make sense of it. From the system point of view it is a file definition, an object that can be manipulated by copying or deleting like any other disk file.

Index — The part of the database that provides a quick way of looking up individual record data. Superbase stores data in 256 byte blocks. Each such block has certain control information in the first few bytes, including the file number (its position in the file catalog, 1 to 15), flags to indicate whether the block is for index or record data, and pointers to other blocks. Superbase registers an error, usually "Index mismatch" or "Data mismatch" if it does not find what it expects at any point in this complex structure. The error condition can usually be cured by copying the database with a special utility, issued free with Superbase version 2.

Record — The blocks of data, each 123 bytes long, that comprise the data input into a particular record format. Records larger than a single block are chained together with pointers stored in each block. If this chain is disrupted, for example by a disk corruption, some data may become inaccessible.

Field — One of the items of data that comprise a record. Each record can have up to 127 fields, one of which must be the index key field.

In addition to the database structure described above, Superbase creates certain other files in the course of normal work. These are listed below.

Lists — The 'find' and 'sort' menu options create lists of index keys as ordinary disk files. A list file consists of a number of lines, maximum length 30 characters, each ending with a carriage return. Lists are used as external indexes when retrieving data from a file.

Memos — Memos are sequential files, and can be loaded or prepared with Superscript. Help screens are memos whose names begin with "h" or "h4" or "h8", depending on the version of Superbase. You can copy, rename, or delete memos.

Programs — In the Superbase context, programs means only those programs created with the Superbase 'prog' program editor. They are not editable in BASIC, and BASIC programs cannot be edited in Superbase. All programs are given the suffix ".p" when they are first stored on the disk, but the suffix is not typed when you 'load', 'execute', or 'save' the program. You can copy, rename, or delete programs, but then you must type the ".p" suffix.

Output data files — These are produced with the 'output to' variant of the 'output' command. An 'output to' file consists of selected fields from the records of a database file, arranged as a simple sequential data file. Such files can have many different structures. They are used for database reorganization.

Export files — Export files are created with the 'export' command. They too are sequential files, but no field selection is possible. The contents of each record are added to the disk file in field order. Both output data files and export files can be copied, renamed, or deleted.

Sequential files — The Commodore 128 version of Superbase includes an 'open' command, which opens a sequential file on the disk. Any kind of output you do when a file is open is directed to that file. This makes possible a number of more advanced operations. When the file is closed with 'close', output reverts to normal. Files created with 'open' may be copied, renamed, or deleted.

CAUTION. *Be extremely careful when you use commands that affect disk files. Early versions of Superbase allow you complete freedom to overwrite files with commands such as 'find', 'sort', 'output to', 'memo' and 'export'. 'Find' and 'export' seem to generate the most mistakes. Remember, never use the name of the file format as the name of a list or of any other kind of file. If you do you may have to go through the formatting process from scratch.*

Deleting Records, Files, and Databases

Much confusion exists in this area. People regularly call Precision Software saying that although they have deleted a file it still seems to exist on the database catalog, and even if they manage to remove it from there it persists on the disk. How do we rid ourselves of these troublesome things?

The difficulty is that a database file has three components; all must be deleted correctly to remove the file completely. The components are records, database catalog entry, and file definition on disk.

Records. Use the 'select delete' option to remove all the records in turn. Alternatively, write a five line program to do it for you (see Chapter 6). There are examples elsewhere in this book. You can tell when all the records have gone by the number of records indicator against the file name in the database catalog -- it should be zero. Also, you will get an "End of file" message when you try to flip through the file.

Database catalog entry. When, and only when, all the records have been deleted, ensure that the file is selected, and use the 'select delete' option one more time. The file name will then disappear from the database catalog. Now select another file, or better still reselect the database, and then choose another file.

File definition on disk. The file definition exists on the disk as an entity quite separate from the database itself, which has now been dealt with. You may prefer to leave it there, as you could need it again in the future. But if you want to remove it, select the 'maintain' submenu, and from it option 8, 'other'. Now type in:

```
s0:filename <return>
```

where "filename" is the file definition you want to get rid of. After a confirmation check, away it goes. End of story.

File Size and System Design

One of the commonest problems the customer support service at Precision Software has to deal with is the "Disk Full" situation. In this section I want to provide some guidelines to help you design a system that will not encounter this error, which can be very frustrating and time consuming.

First, try to figure out the average size of your records. The rules for this are given in the technical appendix to the Superbase Manual. Here's an example.

In a file of customers, the average number of characters is thought to be around 150. This includes the customer code, name, address, city, county, and postcode, as well as telephone number, account balance, and comments. We also allow one separator in

between each field. Since Superbase works with blocks of 123 characters, each record will require two blocks, the equivalent of one Commodore DOS block as used in the "Blocks free" calculation.

On a disk with 150K of space, equivalent to 600 blocks, there will be room for a maximum of 600 records. But this figure must be reduced, as we have not allowed for storage of index data.

The average length of the customer code is eight characters. This allows 32 index keys per Commodore block of 256 bytes. But only 65% of each block is used, to allow room for expansion. So there will be a maximum of 20 index keys per block. Working backwards from our estimate of 600 records, we can see that 30 256-byte blocks will be needed to hold the index data: divide 30 by 4 (number of blocks per 1K) to get 7.5K. But this is only the bottom level of the index.

The higher levels of the index, which is a "tree" structure of branching pointers and data, follow the same rules as the lowest level. To point to our 30 index blocks we will need only two more blocks to hold the 30 index entries at the rate of 20 per block maximum. Then we will need one more block at the top level holding just the pointers to the two below it.

Top level: 2 entries, 1 block: 0.25K

Middle level: 30 entries, 2 blocks: 0.5K

Lowest level: 600 entries, 30 blocks: 7.5K

So the total space required for the index is 7.5+0.5+0.25=8.25K. Add this to the 150K we assumed for the 600 records to begin with, and we have about 159K of disk space accounted for.

Now give yourself room for manoeuvre. Reduce the number of records by 20%. This gives a maximum of 480 records for the file.

Here is a summary of the procedure:

1. Calculate average record length in characters. Refer to technical appendix in Manual.
2. Divide by 123 and round up to nearest integer.
3. Divide by 2 to get number of CBM DOS blocks per record (a).
4. Assign a proportion of the available disk space in K. Multiply by 4 to get number of 256 byte blocks (b).
5. Divide b by a. Result is maximum number of records (c).

   Now adjust for the index.

6. Calculate average length of index key.

7. Divide into 256 to get number of keys possible per CBM DOS block.
8. Take 65% of this figure and round down to nearest integer. This is number of index entries per block (d).
9. Divide c by d (rounding up) to get number of blocks for lowest level of index (e), and divide by 4 to get number of K required.
10. Divide e by d for next level of index. Divide by 4 as above. Repeat this step until the result is 1.
11. Add results of last two steps to get total number of K required for index.
12. Add this to number of K assigned for records in step 4. If you've exceeded the disk space, assign less space and go back to step 4.
13. Reduce maximum number of records by 20%.

You should repeat this calculation for each file in the database.

The reason for the final step, reducing the number of records by a substantial percentage, is that you must conserve disk space for creating lists and data files. If you fill the disk right up, you may find it hard to reorganize the database to recover from the errors a "disk full" situation can engender.

Volume of Transactions

Some databases are fundamentally static. Once the data has been entered it changes little. Others involve the addition of new records, such as invoices, on a daily basis. Whichever type yours is, it helps to know how often you will run out of disk space. The maximum number of records is one relevant piece of information. The frequency of transactions is the other.

You have calculated the number of records you expect to go into your database. This may have involved calculating the number of new records that will be added every week or month. Do this now if you've not already done so.

You now know how often you will need a new disk.

Multi-volume Systems

If you realize as a result of your calculations that a single disk system will be inadequate, you have two alternatives: opt for a bigger capacity disk drive such as the 1Mb SFD1001 or the 10Mb ST100 (or Sider for Apple users); or plan a multi-volume system for the smaller drive.

There are two basic types of multi-volume system: those based on the structure of the data, and those based on time.

A good example of a multi-volume data structure based system is the London telephone book. Its four volumes divide up subscribers

into sections A-D, E-K, L-R, and S-Z. The system allows room for growth, and is easy to understand. If you use alphabetic keys you can adopt a similar system. If these sections are still too large, subdivide them. There are floppy disk systems in the world with 30 or 40 volumes, and they work fine.

A time based system is a little more complex. Typically, such a system will be an accounting system in one form or another. If so, try to use one disk per accounting period. Then you can set up end of month procedures based on whole files without having to do complicated date based searches and sorts.

In this kind of system, it's important to maximize efficiency and ensure accuracy. Almost by definition, control information must be carried forward from one period to the next. Here are some tips for achieving a degree of operational efficiency.

1. Set up a disk with all the databases, file formats and any other files you want on it. Ensure that each file that will have data carried forward has a "dummy" text field two characters long as the last field in the format. Do not enter any record data.

2. Use the single drive backup option to create copies. Create a year's supply, plus one or two spares and one to start off next year.

3. At the end of each period, place any data to be carried forward into a sequential file. You could do this with 'output to', using a single control record with a special key which you put into a one key list with 'memo'. Or it could be a longer file consisting of full account information, again created with 'output to'.

4. Use a utility to copy the sequential file with the control data (you may need more than one for different files) from the current disk to next month's disk. This is done outside Superbase.

5. Insert the new disk, select the database and the relevant file, and 'import' the control data from the 'output to' file. Repeat for each file. The dummy text field holds the carriage return that Superbase places in the 'output to' file as a record separator.

Before you do this, be sure you fully understand the 'output to' and 'import' operations. These are discussed in the section on reorganizing the database in Chapter 10.

CHAPTER 4
BASIC MENU OPERATIONS

Adding Records to the File

The basic Superbase data entry routine is the 'enter' option on Menu 1. There is little to be said about it that is not already explained in the manual. The option has only one purpose, to add new records to the selected database file, and few variations of method during use are possible. However, 'enter' is not the only way to add new records to a Superbase file: 'select add' ('select a' is the more common abbreviation) and 'import' are equally relevant, not to mention the possibilities conferred by the Superbase programming command 'ask'.

Replicating Records: 'select a'

The difference between 'enter' and 'select a' is that the latter starts from the basis of an existing record whereas the former always begins with an empty format. With 'select a', you first call up any record using one of the options on the 'select' submenu. When you press 'a' for 'select a', the cursor appears in the first field of the current screen, which is usually screen 1. Now you can choose to change the contents of any of the fields, except for result, calendar, or replica fields. You can edit the record in the normal way, moving from field to field and from screen to screen. At any time, you can store it. This means that by simply making the contents of the key field unique, you can very quickly produce a duplicate of a record.

When you are entering data in batches, it may pay you to sort it into a rough order first, so that, for example, all the new records for a name and address file that share the same surname can be input together. Then, if you are using index keys based on name, you can use 'select a' to quickly modify the key for each new record. If you can also manage to pre-sort the records so that elements of the address can be carried over from one record to the next, so much the better. With a little thought, you should be able to see ways of saving time in any application that requires frequent entry of new records to a file.

The Problem of Index Key Sequences

I have stressed the importance of avoiding duplicate index keys except for special applications. The recommended method for preserving uniqueness in the index key field when there are a number of identical origins for the key, as in a name and address file that contains many people called Smith, is to add a numeric suffix to the alphabetic root:

```
smith01
smith02
smith03
smith04
smith05
```

So far so simple. The problem arises when there are, say, thirty-four Smiths and you cannot remember which number to give the Smith whose record must be added next. Here are some guidelines to help you over the difficulty.

1. Keep a master printout of the file in a readily accessible form. Reprint it often enough to prevent it becoming hopelessly out of date.

2. When you add a record that increases the number in a key sequence, say from 'smith59' to 'smith60', write the new number on the printout, and cross out the previous highest.

3. If there isn't a printout, use Superbase's own logic to find out what you want. If you tell Superbase to find a record by looking up the index key -- the 'select key' option -- it calls up the record whose key most nearly matches what you enter. If it can't find a match, it calls up the next record in sequence. When doing this, Superbase uses a strict letter by letter comparison method. You can use this to get quickly to the last record in a sequence:

   A. Use 'select k'. When prompted for a key, enter a string of letters that just misses the last record in the sequence you want. For example, you want 'smith', so enter 'smj'. Even if Superbase finds a name beginning 'smj', which is unlikely, you know that the record on the screen comes immediately after the last record beginning 'smi'.

   B. Now use 'select previous' to call up the previous record in the sequence. Most often this will be the one that shows the highest existing number in the key sequence.

   C. Use 'select a' or 'enter' to add the new record after making a note of the required sequence number.

   You may need to enter a string longer than two or three letters. In the above example, all records beginning 'smi' come before the record beginning 'smj'. So 'smithson' and 'smithers' will appear when you use 'select p', before you get to 'smith34'. However, the longest string you would need to enter in this example to be sure of finding 'smith34' as the next previous record would be 'smith9'.

Converting from Duplicate Keys to Unique Keys

Despite the cautionary notices plastered all over the manual, several people end up setting up duplicate key files. Most of them regret it, and would change to unique keys if they could. This is not too hard, and involves a combination of manual labour and database reorganization tricks. I discuss it in detail in the section on reorganization in Chapter 10.

Importing Data

The Superbase 'import' operation involves taking data from a simple sequential disk file and storing it in a previously created Superbase database file.

It's best to think of importing as a simple way of automating the intake of data. Whether you are converting from another more primitive database or simply shovelling records from one Superbase file to another, 'import' is extremely useful.

The key thing to remember is that the file format and the arrangement of data in the disk file must correspond exactly. If they do not, the operation will become unsynchronized and fail, usually with an "Invalid FMS parameter" error message.

The data in the disk file must come in the same order as the order of the fields in the file format. For example, the ubiquitous name and address file might look like this:

```
code      <                    >
name      <                    >
street    <                    >
city      <                    >
state     <                    >
zip       <                    >
```

The disk file must follow the same order:

```
jones01
Barry Jones
234 Main Street
Winesburg
OH
34567
keller04
Wayne Keller
1289 Rose Avenue
Denver
CO
78912
```

If the data looked like this it could not be imported:

```
Barry Jones
234 Main Street
Winesburg
OH 34567
Wayne Keller
1289 Rose Avenue
Denver
CO 78912
```

The absence of a key and the use of a single line for both state and zip information leaves only four lines per address, whereas the file format is set up for six. Two techniques for overcoming this kind of problem are:

1. Insert blank lines at appropriate places in the data file. You may have to go through the intermediate step of importing into a file that matches the disk data file, and using 'output to' to insert spaces in between existing fields while creating a new disk data file. Now you can import into a format of any design.

2. Remember to make your new file a duplicate key file. After importing the data, print out all the key field entries. Go through the list writing in any changes needed to create unique keys. Now edit the file using 'select a' to replicate the record data with a new index key where necessary. Go back to the originals and delete them. Print out a new list and check your work. Finally, use the 'format' option to answer 'n' to the "Allow duplicate keys" question, thus converting the file to the preferable format.

A point to remember is that you can import directly into a result field, and even set one to a false value (that differs from what would be computed by the formula). You would have to make Superbase to compute the field to obtain a correct result.

PROGRAMMER'S TIP. *Users can become frustrated by the time taken to store a new record. Batch data entry can be programmed. You use the 'ask' command to capture data for each field in turn, storing it in string and numeric variable arrays. This allows input validation. When a number of records have been entered, the program loops through the arrays assigning the data to the correct field names, and stores each record with the 'store' command.*

Reviewing the File

Once you have a number of records in your file, you will undoubtedly want to look at them, if only to check that the details are correct. Superbase provides a menu of options which allows you flip through your records quickly and easily. This is the 'select' submenu. Its options are:

Command	*Abbrev.*	*Function*
key	k	look up by key field
current	c	show current record
next	n	show next in key sequence
last	l	show last in key sequence
previous	p	show previous in key sequence
first	f	show first in key sequence
match	m	show records to match values
output	o	dump current record to printer/screen
add	a	create new record from current

replace	r	edit current record and store
delete	d	remove current record from file

First let's look at the 'next/last/previous/first' group. Using these is like flipping through a card index box. Repeated pressing of 'n' or 'p' moves you forward or backward through the file; 'f' or 'l' gives you the first or last record in the file respectively. If you try 'n' on the last or 'p' on the first record, Superbase shows the message "End of file".

The 'key' option provides a way of jumping to any place in the file, by typing in an index key field, or just a part of one. You can go to 'smith', back to 'adams', and on again to 'mellors', each time pressing 'k' and typing in the key when prompted "Enter key". The partial key entry feature helps you by allowing quick movement near to a record whose full key you cannot remember, or which has a long key that you don't want to type in full. I always find it useful to jump to the first of the keys beginning with a given letter of the alphabet just by typing in "j" or "q" or "v" or whatever I want. Then I use 'n' to flip forwards until I find the record I need.

You will have noticed several references already to the "current record". This is an important Superbase concept. Every file has its current record, except when the file has just been selected and no record has been retrieved from it. Whenever one of the options I've just discussed is used, a record is displayed on the screen. This is the current record. It remains the current record until another selection replaces it. This means that you can go back to the main menu, do calculations, write a memo, even select another file before returning to the first one, and the current record will still be there. You summon the current record at any time with 'select c'; this instantly displays it on the screen. The concept of the current record becomes very important in Superbase programs that need to call up records and process them one by one.

The 'select match' option is intended to let you call up a selection of records from the file. As the manual explains, you type in the values you want onto a blank record format, and Superbase searches the file for the ones that match your specifications, showing them one at a time. You call up the next matching record by pressing 'm' again, a system that allows you to use any of the other commands in the interim. Some people don't realize that this means that you must force Superbase to terminate one sequence of matching records before you start another. Do this by pressing 'l' to get to the end of the file. Then when you press 'm' you will see a blank screen ready for you to type values onto.

In my experience, 'select o' is infrequently used. The most natural way to use it is to combine it with 'select m' to call up a sequence of records, dumping the contents of the ones you want onto printout. To do this you must first switch output from the screen to the printer -- do this by typing 'print' while at the Menu 1 screen, and pressing return. Type 'display' the same way after you've finished to switch output back to the screen.

I have already dealt with 'select a', the command for creating new records out of old. The 'select replace' command is your main record editing command. To use it, first call up the record you want to edit, using any of the commands available. Then press 'r', and the cursor will be positioned in the first field of the current screen that can be edited. You cannot change the index key field with 'select r' -- this safeguards the record sequence. Nor can you edit a result, calendar or replica field. When you've finished, end the command as the manual says (it varies from Commodore to Apple), and the record will be replaced in the file.

The last command on the submenu is the command for deleting a record from the file. As with replace, call up the record you want to delete first, using any of the available commands. Then press 'd'. Superbase asks you to confirm the removal of the record you are looking at. If you press anything other than 'y' the operation is cancelled. If you go ahead, the record is deleted, and Superbase shows the next record in the key sequence on the screen.

This is fine as long as you don't want to remove a whole file, or a group of records. That could be very time consuming. Fortunately, an automatic record deletion routine is one of the easiest Superbase programs there is, and it's covered fully in Chapter 6.

Searching the File; the "list" Concept

For some people, the concepts of searching and sorting are a little blurred. Since they are both extremely important in Superbase, it is worth spending some time on them before getting into the details of the former.

When we search, we always search for something. This is true in Superbase too. One of the great advantages of a database is its ability to hold large quantities of data and yet allow you access to it in useful ways. In practice, "useful ways" means a lot more than just making printouts of the files. People want to extract groups of records from the database by issuing commands, such as "find the vice presidents of sales and marketing in the states of California and New York", or the more homely "find the albums on which Stevie Wonder plays with Aretha Franklin". It's not a coincidence that I repeated the word "find"; 'find' is the Superbase option that searches the database, and extracts the records that match the values you input. In fact, 'find' finds a "list" of the records that match, as I'll explain momentarily.

Sorting is quite different from searching. You must be careful not to let the secondary meaning of the phrase "to sort out", which indeed implies an element of searching and extracting, become confused with the use of 'sort' as a computing term. I go into more detail about sorting in the next section, but to distinguish it from searching, we can say that sorting is about rearranging the order of records, usually just prior to printing them. Sorting does not involve searching; it's a quite separate activity.

The "list" of Records

Back to searching. I mentioned a "list". The concept of the list is absolutely central to Superbase's way of doing things. You won't be able to use the full power of Superbase if you don't understand how lists work.

First of all, what is a list? It's just as the name implies, a list of items. Each item is an index key entry, such as "smith01" or "jones99", from a database file. Lists are created by Superbase as the end result of the 'find' option. They take the form of a disk file, and so you can easily copy, rename, or delete them. A list can be any length up to the whole number of records in the file on which it is based.

How do you refer to a list? Superbase gives every list you make the same name, usually "hlist" or "h8list" (it varies slightly from version to version), unless you give it another name before you start or rename it afterwards. The list with this name is known as the "default list". When you refer to it with any of the commands listed below, you can use just the two double quotation marks, "".

How do you use lists? A list is a subgroup of the records in a file, in the form of the index keys of selected records. You can use the list to process the subgroup in several different ways. Here is a table of options that shows the range.

Option	*Activity*
output	print or display details
output to	create disk file for word processing
sort	rearrange record order for output
batch	automatic updating of records
detail	output in special report format
export	dump record contents into disk file
select from	access records one at a time for processing

How does a list work? Each item in the list is the index key entry for a record in the database file. When a list is used with one of the options above, Superbase reads one key from the list, and then uses it to find the corresponding record in the file (except for 'sort' -- see next section), making that record the current record. When that record has been processed, Superbase reads the next key, and so on to the end of the list. So I sometimes refer to the list as a temporary index, which is what it is, although different from the internal database index described earlier.

A list is valid as long as no records are added to or deleted from the file. Also, although I have mentioned only 'find' as a way of creating a list, there are other ways. You can type one in yourself with the 'memo' option, or use a word processor such as Superscript to produce one. If you do this, you must be sure that every key on the list is valid, or there will be errors when you go to use it.

The effectiveness of your searching strategies depends on two things: your understanding of how to specify search values, and the structure of the record format you want to search. These are interrelated, and the best way to an understanding of the latter is through a look at the former.

Ways of Searching the Database

When you select the 'find' option from Menu 1, Superbase presents you with the first screen of the currently selected file, so that you can type in the values, or criteria as the manual calls them, into the fields to which they apply. The record format is shown with square bracket field end markers to distinguish it from normal presentation.

```
code        [                    ]
name        [                    ]
street      [                    ]
city        [                    ]
state       [                    ]
zip         [                    ]
```

Exact Match

To obtain a match that is exact, except for case differences which Superbase does not attend to, you include an equals sign:

```
code        [                    ]
name        [=John Smith         ]
street      [                    ]
city        [=London             ]
state       [                    ]
zip         [                    ]
```

This will find only records where [name] has the value 'John Smith' and [city] has the value 'London'. If anything precedes or follows a criterion used like this with an equals sign, there is no match. So, 'Greater London' would fail, as would 'Mr John Smith'.

Sliding Match

If you want all Smiths, without regard to any of the other contents of the field, omit the equals sign:

```
code        [                    ]
name        [smith               ]
street      [                    ]
city        [                    ]
state       [                    ]
zip         [                    ]
```

This will come up with all the 'smiths' in the file. It will also find Smithers, Smithson, Ladysmith, Hughes-Smith, and Angus McSmith.

Pattern Matching Characters

The characters '*' and '?' can be used to match any number and single characters respectively. You can use '?' to help define a single word in a field:

```
code        [                    ]
name        [                    ]
street      [? Second ?          ]
city        [                    ]
state       [                    ]
zip         [                    ]
```

The string is still matched wherever it occurs in the field, but we have insisted on finding a space at each end of the word. The following instances would be matched:

123 Second Avenue
56 Second Street
Forty Second Street Plaza

A more common use of the '*' character is with the equals sign. We can use the '*' only at the end of a string of characters, to indicate that we don't care how many characters come after it. This allows criteria like this:

```
code        [=c*                ]
name        [                   ]
street      [                   ]
city        [                   ]
state       [                   ]
zip         [=069*              ]
```

Quite simply, this finds all the codes beginning with 'c', where the zip code also begins '069'. Note that by substituting '#c*' we would find all codes except those beginning with 'c'. The '#' character denotes exclusion of the criterion from the search.

Sliding Search Across Fields

You don't have to restrict the search to a single field. This is most useful where long text fields are in use, and you want to find a keyword which could occur in any one of a number of fields:

```
comment 1   [whales-                                ]
comment 2   [                                       ]
comment 3   [                                       ]
status      [*     ]
```

The search will match records where 'whales' occurs in any of the three fields. I have used the same example to illustrate that the '*' character has another purpose: to terminate a sliding search. You might want Superbase to make comparisons in only three fields out of the maximum 127, saving time noticeably in a large file. Placing '*' in a field on its own has the desired effect.

And/or Logic Comparisons

You have seen that it is possible to enter criteria into more than one field; in fact, you can place one in every field in a record format. You can also ask for more than one value to be accepted in a field:

```
code        [=a*/=b*/=c*        ]
name        [                   ]
street      [                   ]
city        [London/Paris/Rome  ]
state       [                   ]
zip         [                   ]
```

Here we are using the special operator to indicate alternatives, '/'. We are asking for all records where the code begins with 'a' OR 'b' OR 'c', AND the city is 'London' OR 'Paris' OR 'Rome'. We can also ask for a range of values to be accepted. Suppose the record format had an [age] field in it; we could request all records where the age was under 50 and above 30 like this:

```
        age       [>30&<50 ]
```

Note two things: the use of '&' to denote 'AND', and the use of the comparative operators '>' and '<', which you can use for text fields as well as for number fields (in text fields this sort of comparison is done on a letter by letter basis). If we actually wanted the 30 and 50 year olds as well, we would have to change the criterion slightly:

```
        age       [>29&<51 ]
```

This should illustrate the need for care when specifying the criteria for a search. This is especially true when it comes to dates. The criterion for all records for October 1985 is as follows:

```
        >30SEP85&<01NOV35
```

October itself is not referred to. I wonder how many readers noticed that this particular criterion is too big for its field? Date fields are not supposed to be longer than 11 characters. In practice, many criteria are too long for the field; naturally Superbase has an answer: the rather jargonish "delayed request character". On Commodore systems this is the back-arrow key. You enter it into the field instead of the criterion, like this:

```
        city      [←                  ]
```

When you press shift/return or Control-R to start the search, Superbase prompts for the criterion on the command line. On an 80 column system you then have room for about 150 characters of criterion -- more than enough for any sensible request.

Avoiding Meaningless Requests

The syntax of search criteria is like a miniature language within Superbase. As with any language, it is possible to make meaningless utterances. For example, the criterion '=Smith&=Jones' is no good -- a field value cannot be simultaneously exactly equal to more than one thing.

Superbase does not have any priorities in its queries. You cannot use parentheses to insist that one set of comparisons be done before another is looked at. So, the criterion 'Paris/London&New York' is meaningless. Is it supposed to be 'Paris' OR 'London and New York', or 'Paris or London' AND 'New York'? Superbase can't tell.

Designing the Format to Allow Effective Searching

This leads inexorably to the realization that the record format must be structured to take into account the kinds of search requests that will be made.

Suppose a rather different record format where the city is not part of a name and address, but can be any combination of London, Paris and New York. If you set up the field like this you could not search for all possible combinations:

```
city      <London Paris New York        >
```

You would soon run into difficulties like the ones described just above. Instead, the record must be set up with a separate field for each possible category, with an indicator such as 'y' or 'n' in it to show whether the category applies:

```
London    <y    >
Paris     <y    >
New York  <n    >
```

Now you can search for any combination:

```
London    [=y   ]
Paris     [=n   ]
New York  [=y/=n]
```

The search will find all records where [London] is 'y' AND [Paris] is 'n' AND [New York] is 'y' OR 'n'. The third term would exclude records where [New York] was blank.

An alternative to multiple fields, one for each search category, is a special purpose selection code field. Such a field must have a predictable structure to be effective. It could be a constant field, looking like this:

```
Selection <xxxxx>
```

Each one of the 'x' characters can be replaced by a letter, say between 'a' and 'e', so the field contents could range from <aaaaa> to <eeeee>, with of course any of the 'x' characters left unchanged, as in <axxbd> or <xcxee>. Each position and each letter has its own significance, purely for the purposes of searching. Now, provided you follow the rules imposed by the structure, you can extract many different combinations:

```
??a??/??d??/???e?
```

This finds 'a' or 'd' in the third position, or 'e' in the fourth position.

```
bbbcc
```

This finds 'b' in the first three positions AND 'c' in the last two.

```
a&b
```

This finds any record with an 'a' and a 'b' anywhere in the field. With careful forethought, a very sophisticated system can be set up using this technique.

Searching for Dates

One example of how to specify a search for all dates for a particular month has been given above. Basically, you use the greater than and less than operators, '>' and '<', to indicate the boundaries of the search. So, '>31DEC84' will find all dates for 1985; '<01NOV85' finds all dates before November 1985; '>31DEC84&<01NOV85' finds all dates between January 1st and October 31st, 1985; and so on. You can also use the equals sign operator for single dates, and the OR operator '/' where it's meaningful.

Often users want to input a date in a program, using the 'ask' stauement, and then base a 'find' operation on it. This is easy, but requires some understanding of variables and how to construct the equivalent of the search criteria screen in a program line (see Chapters 7 and 10).

Sorting: Rearranging the File

Searching with 'find' gives you the ability to extract any group of records from a database file. It produces a list of the index keys for the records that match the criteria you enter for the search. The list itself is still in the order of the basic file, that is in alphabetic index key order. Often, you will want to print your records in some other order. I have given zip or post code order as one example. In practice, most files need to be printed in a sorted order different from the basic index key order at some time or other. Here are some examples of typical rearrangements achieved by sorting:

File	*Sorted By*
names and addresses	zip/post code
customer accounts	balance
invoices	customer AND date
products	product type AND quantity on hand
personnel	date (i.e. seniority)
library	author AND title

You can see that it is possible to sort by more than one field at once. In fact, Superbase allows you to sort by as many field names as you can fit on a command line; this varies between 40 and 80

column versions of the program, and also depends on how long the field names are. The maximum for a 40 column system is 34 fields, with a limit of 45 for the 80 column screen. Usually you need no more than four or five at once, and often just one or two is adequate.

The 'sort' option is selected from Menu 2. You have to type in the names of the fields you want Superbase to use in the sort. Superbase then does the sort, and produces a "list" of the index keys to the records in the new order. The list produced by 'sort' is exactly the same kind of thing as the list produced by 'find', the only difference being that it is not in the basic index key order of the database file. The change in the order of the keys is illustrated by this example:

Key	*Sort Field*	*Sorted List*	*Output order*
adams	Wyoming	charles	Alabama
baker	Connecticut	baker	Connecticut
charles	Alabama	douglas	Nebraska
douglas	Nebraska	adams	Wyoming

When you select 'sort' you must also specify whether you want to sort all the records in the file, or just the records whose keys are in an existing list, which you have previously created with 'find'. This is the meaning of the 'all/from "list"' prompt that comes up when you select the option.

CAUTION. *'All' and 'from "list"' are alternatives. You should not type in 'all the records from "hlist"', even though Superbase would manage to function by taking the last entry, 'from "hlist"', as the intended one.*

Here are some examples of sort commands typed in after the 'sort' option has been selected:

```
all on [zip] to "sortlist"

all D- on [balance] to "custlist"

from "monthlist" on [customer][date] to "chaselist"

all on [product-type][quantity] to "stocklist"

from "" on [date]

all on [author][title] to "booklist"
```

The second example introduces the idea of the descending sort. The normal index key order, which I have referred to as alphabetic, goes from "0" to "Z"; as far as sorting is concerned, this is the ascending order. But if you want to see your customer records in order of balance, you probably want to see the largest balance first. Since this is an order based on decreasing magnitude, we call it descending order. You obtain a descending sort by placing the characters 'D-' before the list of sort fields. The term

"descending", also applies to dates, in which descending order means from most recent to earliest, and to text, in which it means from "Z" through "z" to "0" (actually the sorting order, for both ascending and descending sorts, is based on the ASCII code, and extends to include all printable characters including the space).

Whichever order you choose, the index key itself should not be included as one of the sort fields (unless it comes before the final field in the line), since Superbase automatically uses the key as the final field, thus ensuring that index key order, or its descending converse, governs the final result. If, in example two, there were three customers with balances the same, the records would be in the descending sorted order:

Balance	*Index Key*
2300	smithson
2300	smithers
2300	smith
1900	adams

Normally Superbase can only do either an ascending or a descending sort. It is possible to set up special result fields in the record format to produce reciprocals or complements of data elements so that an apparently ascending sort using such a field in fact permits output in descending order. I go into this in more detail in Chapter 7.

The fifth example in the list above shows how to refer to the default list in a command line. Use "". The same example also shows that you can omit the name of the destination list, the one that holds the index keys in their new order. If you do this, the new list will replace the old one under the same default name, e.g. "h8list".

Sorting can be time consuming. To achieve its ends, Superbase has to read in a group of records, sort them, write the index keys to a temporary file on the disk, read in the next group, sort it, merge it with the first group into the temporary disk file, and so on. If you want to keep sorting time down, reduce the number of fields used to the minimum, and if possible specify the minimum number of characters -- here is an example of how to specify that six characters only are to be used when sorting:

```
all on &6[description]
```

Automatic Updating

Updating records can be done in two ways: either with the 'select replace' command, which we've already looked at, or with the 'batch' command, which we consider here.

'Select r' is used for amending the details of individual records. 'Batch', on the other hand, either processes the whole file or uses a "list" created with 'find'. As with 'sort', when you select this option from Menu 2, Superbase presents you with the prompt:

```
all/from "list" (item list ....)
```

You type either 'all' if you want to update a whole file, or 'from "listname"' if a subgroup is the target. Then you must specify the expressions that determine how the updating is to be done. Basically, an expression consists of a field name on the left of an equals sign, with an assignment on the right. It's similar to a BASIC language 'let' instruction:

```
all [amount]=[amount]+1
```

This command reads through the whole file record by record. For each record, it adds one to the [amount] field, and stores the record. You can use the same kind of arithmetic to achieve percentage changes as in the result field examples earlier on:

```
all [amount]=[amount]*1.1
```

More interestingly, you can include more than one updating expression in a single command line. However, you may not have any given field on the left of the equals sign more than once in the line.

The multiple assignment technique allows you to maintain running totals in a record format, when the fields have been set up properly to begin with:

```
new amount    <          >
running total <          >
```

During data entry, only the [amount] field receives any input. The objective is to read the file, add the contents of [amount] to [total], and reset [amount] to zero. The 'batch' command line to do this is as follows:

```
all [total]=[total]+[amount];[amount]=0
```

In a large file with few updates due, you would save time by first doing a 'find' to produce a list of all the records where [amount] was greater than zero ('>0'), and telling 'batch' to process the list instead of all the records.

In the example above, notice the use of the semicolon to separate the expressions. This is essential.

'Batch' can be used for many other activities, including totalling, counting, calculating maximum and minimum values, and doing conditional updating.

Doing Calculations

Since the overall subject at this point is Setting Up a System, it should be useful to have some idea of what the 'calc' function, option 5 on Menu 1, is for.

It has no practical relation to the result field formulas we looked at earlier, though they do in fact use some of the same internal Superbase code. 'Calc' is most often used to do quick arithmetic, using BASIC operators, as in these examples, which you type in after selecting 'calc' from Menu 1:

```
23*45   <return>

3456*.15   <return>

144/17   <return>
```

The result is displayed on the screen until you press return.

You can also use 'calc' with the fields of the current record. So you could type, for example:

```
[amount]/7   <return>
```

The result is displayed, but the contents of the field are unchanged. On the other hand:

```
[amount]=[amount]/7   <return>
```

inserts the result of the calculation into the [amount] field itself. To see the result, you would use the 'select c' option to display the current record.

Output

This is the final objective of the whole database system. We set up file formats, enter data, update it, review it, extract lists from it and sort them, only so that we can, when all is done, produce some output.

Superbase is very good at this end of the system. It's best to think of output as having three directions or modes: printer, screen or disk (only one at once). The last one, disk output, is for word processor data files, and is mainly concerned with listing fields in a certain order. The other two, printer and screen, are more frequently needed. Most of the points I shall make in this section apply to both these types of output, but there are often some small differences between them, even though I don't always give details unless it's important.

When you want to switch output from screen to printer or vice versa, you must use one of the direct commands 'display' or 'print'. You type these in from Menu 1, pressing return at the end. Whichever was typed last determines the direction of all output until the other is typed.

The 'output' option is the fourth on Menu 1. When you select it, you see the same prompt as for 'batch':

```
all/from "list" (item list)
```

As with the other commands, you specify either all the records or the name of a list that you previously created with 'find'. Then you go on to specify the items you wish to output. Here is a table indicating the kinds of element that can be used with this option:

Item	*Example*
field	[name] [street] [city] [amount] [date]
text	"Telephone:" "Current status:"
formulas	[amount]*[quantity] [date]+90 int([amount]/1000)
BASIC variables expressions and functions	a$ amt asc(x$)

You simply type in the names of fields or any of the other elements you want to output and Superbase produces a list. A typical 'output' command line looks like this:

```
all [name][street][city]"Phone:"[telephone]
```

The resulting list will be:

```
John Jones    21 Main Street     Winesburg     Phone: 503 456 7891
Simon Poole   3456 Ocean Blvd    San Jose      Phone: 715 685 2345
Mary Valdez   1212 Pine          Laguna Beach  Phone: 912 234 4564
etc.
```

These are the characteristics of the basic output list:

1. The full length of fields as specified in the format is output. Trailing spaces are not removed from text fields. This produces regular columnar output, as above.

2. Numeric fields are output with the full numeric format of nine character positions before the decimal point, and two after. There is one extra position for the sign, and one for the decimal point itself.

3. Every kind of element is output with one space after it. Always allow for this when calculating the expected position of an output item.

4. The elements of the 'output' command line appear once for every record processed.

5. Date fields are output at a length of seven characters.

6. There is no way to perform tests on the records as they are output.

This kind of output is often all that is needed. However, many users want to produce more carefully formatted output, and to help them Superbase provides a number of formatting commands to perform the following functions:

Command	*Function*
&	Truncates text
&5,2	Formats number, e.g. as 99999.99
@x	Positions item at column x
@x,y	Positions item at column x, row y

Various combinations of these commands, and indeed of a number of subtler variations on them, all of which are listed in the Manual, allow output items to be very flexibly manipulated. A typical command line could be:

```
all@5[name]@25&5,2[balance]@40[telephone]@5[street]
@5[city]@5&[state][zip]
```

The resulting output would be:

```
5                   25             40
:-------------------:--------------:-----------------

John Jones               678.89     503 456 7891
21 Main Street
Winesburg
OH 34567
Simon Poole             1234.00     715 685 2345
3456 Ocean Blvd
San Jose
CA 91232
Mary Valdez              456.99     912 234 4564
1212 Pine
Laguna Beach
CA 9345b

etc.
```

There are a number of things to notice about this example. I have included a ruler line to illustrate the effect of the positioning commands. Each of the name and address items is positioned at the same column. This ensures a neat format, but you must take care that the items are mentioned in the command line in the order in which the printer will physically deal with them -- it cannot do the address and then go back up four lines to print the balance and phone number. The numeric field, [balance], is positioned at

column 25; it is printed right aligned, so the 5,2 format results in leading spaces. The [state] field is truncated of its spaces, and followed immediately by the [zip] field without a positioning command. Lastly, there are no blank lines between the records. To obtain one, you place the additional formatting command '@1,0' at the end of the command line.

The example assumes that the [name] field is no more than 18 characters long. 18 characters printed at column 5 brings us to column 23; add one for the space at the end of the field, and the next field starts at 25. But what if the [name] field was longer than 18? Say it was 20. You would see this result:

```
5                   25             40
:-------------------:--------------:-----------------

John Jones
                        678.89     503 456 7891
21 Main Street
Winesburg
OH 34567
Simon Poole
                       1234.00     715 685 2345
3456 Ocean Blvd
San Jose
CA 91232
Mary Valdez
                        456.99     912 234 4564
1212 Pine
Laguna Beach
CA 9345b

etc.
```

Superbase tries to print [balance] at column 25 on the same line as [name], fails because that column has already been occupied by the tail end of [name], and prints [balance] at column 25 on the next line, disrupting the format. If you find fields going adrift in a printout, this is almost certainly the reason.

Output 'across' or 'down'

The examples we've looked so far have all been of output 'across'. Superbase starts outputting the details of each new record on a new line. This is the most common form of output, and this is what is set when Superbase starts up. However, there is another form of output, 'down', which is set either by typing it on the command line or by incorporating it into the main output command line. Output 'down' means that Superbase starts every new record on a new page: "page by page" output.

Page by Page Output

It is quite possible to print the details of each record at any

position on the page. You are not restricted to line by line output. A command line to illustrate the extreme of this approach would be:

```
all down @5,1[name]@55,60[balance]
```

We would then see the following:

```
John Jones
.
.
                    (55 lines)
.
.
                                                    678.89
Simon Poole
.
.
                     (55 lines)
.
.
                                                   1234.00
Mary Valdez
.
.
                     (55 lines)
.
.
                                                    456.99
```

The limitation of 'output down' is that you can only put one item on each line, and in many cases 'output across' is in fact preferable for producing documents such as invoices.

Output Parameters

There are a number of direct commands for altering the parameters of the 'output' function. Each one is typed on the command line followed by the appropriate value. The values in the examples below are those set when Superbase starts up.

Page length:	plen 66	The 'plen' command sets the maximum number of lines on the paper at six lines per inch. The default is for 11 inch paper. Pre-printed stationery is often shorter than 11 inches.
Text length:	tlen 60	This command limits the number of lines allowed on the paper, effectively determining the bottom margin. You determine the top margin by aligning the top of the paper physically.
Left margin:	lmarg 1	The value given sets the left margin. You may need to use this if the default

output is crowded against the edge of the paper. If you do, remember to increase the right margin by the same amount or more to avoid compressing the output line length.

Right margin: rmarg 80 — Sets the position of the right hand margin. The maximum value is 255 for the printer, 80 for the screen. This makes "landscape" style printing easy, provided the printer can handle it.

Spacing: space 0 — Sets single, double or triple spacing. Every line is spaced accordingly, so this is no good for separating records grouped as in the example above, although it would be suitable for one line records.

Continuous print: cont 0 — Normally Superbase prints without pausing for a break between pages. To change this, type 'cont 1' from Menu 1.

Linefeed: lfeed 1 — If the printer overprints all lines, use 'lfeed 1' to switch linefeed on; conversely, 'lfeed 0' should eliminate unwanted double spacing.

Totalling

A much desired item, that is unfortunately not possible with the simple 'output' option. If you want to produce totals from the fields that you print, you must use the various commands available for reporting, which I discuss at some length in Chapter 8.

Disk Output

Output to the disk is often required for word processing mail merge operations. (Of course this is not necessary if you own the Commodore 128 version of the program, which integrates in memory with the Superscript word processor from Precision.) Disk output is achieved with a variation of the 'output' command, 'output to'. Basically, you have the option to specify a file name as the destination of the output. All the possibilities of formatting and 'across' or 'down' output are allowed, and you can generally create a data file to suit whatever the word processsor wants. As far as Precision products are concerned, the 'fill' subcommand can be used to specify output compatible with Easy Script or Superscript. First, an example of a command line that inserts commas between the fields:

```
all across to "datafile" &[name]","&[street]","&[city]
"&[state]&[zip]
```

The result would be:

```
          John Jones , 21 Main Street , Winesburg , OH 34567
```

With 'fill' the command line looks like this:

```
          all fill to "datafile" [name][street][city]
          [state][zip]
```

The result would be:

```
          John Jones
          21 Main Street
          Winesburg
          OH
          34567

          Simon Poole
          3456 Ocean Blvd
          San Jose
          CA
          91232

          Mary Valdez
          1212 Pine
          Laguna Beach
          CA
          93456
```

Notice the blank line inserted after each record.

This ends Part I, Setting Up a System. We have covered all the important elements of the Superbase system as they are first encountered by the average user, and indeed by all users as they first learn the system. In the next section we shall build on this basis and see how to create a system that uses simple programs to obtain a significant increase in power and convenience.

PART II

THE AUTOMATED DATABASE

CHAPTER 5
USING THE COMMAND LINE

Beyond the Menus

Many Superbase users are quite content with the menu driven capabilities of the system, and indeed resist the suggestion that they might make things easier for themselves if they were to learn something about programming. The idea of programming apparently terrifies some people, which is why I introduce the more familiar concept of automation first. After all, everyone understands the advantages of an automatic transmission in a car, or an automatic pilot in a plane. Of course, Superbase is itself an example of automation in action. Whenever you add a record to the database, Superbase automatically inserts it in the correct place in the file. And when you use 'find' to search the file, thousands of comparisons may be made automatically. Automation, in fact, is both a purpose and a method.

So, the following pages are not a jump into a swamp of intimidating jargon, but a natural extension of your understanding of Superbase from the point you should have reached if you've read this far -- a fairly good sense of the basics of the system, and of how the main menu options are related and intended to be used. You are going to learn how to make life easier by automating a number of repetitious operations, saving both time and the effort of remembering a lot of tedious detail.

I shall begin with a discussion of the Superbase command line, and how to use BASIC memory variables to advantage. Before getting into the first practical example, which deals with deleting records en masse, I spend some time examining the crucial first step of planning your actions. The deletion example is followed by some advice on managing your programs -- to prevent you accumulating disks full of junk. Then we go through the automation of what I call the "core" activities of the system: searching, updating, sorting, and output. The section ends with a close look at the 'report' functions, which allow several powerful analytical operations on the database.

The Command Line

Some users have no idea that Superbase has a command line. This might be considered a proof of the virtues of the system -- you can run it entirely from the menus -- or a tiresome fault -- why on earth aren't users informed about it with a simple cursor prompt? On balance, and after much experience of educating incredulous owners, I incline to the latter view. The command line is much too valuable to be left as a buried treasure. So our first job is to excavate it.

What is "the command line"? When you're looking at the normal Superbase screen, either Menu 1 or Menu 2, you cannot tell from anything on the screen that it is possible to type anything other than the menu option function keys. In fact, you can type any key

on the keyboard, except a number, and the characters pressed will be displayed on the second line of the screen. By typing a valid Superbase command, such as one of the ones listed on the menus, and pressing return at the end of it, you directly affect the system in some way. The commands are not restricted to the menu options. They can include all the Superbase commands listed in part two of the Programming Section of the manual, and indeed any legal direct BASIC command that Superbase does not disallow. If you type garbage, Superbase will tell you; likewise if you make a mistake in the command -- the "syntax". The only thing you cannot type directly is a number, but you can always tap the space bar once to bring up the command line, and then type the number.

Important Commands

The command line is invaluable. It gives you great control over the state of the system, allowing you to change the parameters for everything from the number of the current disk unit to the contents of a single field in a record. Here are some of the most important commands for operations ranging from database selection to printer margin setting. You should be familiar with these commands if you are going to learn to program the system, so I suggest you read up the summaries in the Manual (Programming Section, part two)

Every command is completed by pressing return.

database	Sets, or logs in to, the current database. Obviously a crucial command. You can set the database on any disk drive that the system can address. This applies to a second disk unit provided Superbase can address it as a separate unit.
database "sales",8,1	Logs in to a database on drive 1 of a dual drive unit.
database "sales",9,0	Logs in to a database on drive 0 of unit 9 -- a second unit.
database "sales"	Logs in to a database on drive 0 of the current unit; in the case of a single floppy, this is obviously the only drive. If you omit the specifier, the system always looks for a database on unit 8, drive 0 -- it does not look at the most recently used drive.

(The above examples are for Commodore drives; for Apple users the logic is the same, but the syntax is different, so refer to your manual.)

directory	The familiar command that shows the disk contents, also available from the 'maintain' submenu, but very often used directly, especially in its abbreviated form (see below). The Apple equivalent is 'cat' or 'catalog'.

Selective views of the directory are possible. If you name your files carefully, you can use the pattern matching characters '*' and '?' to produce partial directory displays. You cannot use 'directory' for this, but must turn to the 'other' option on the 'maintain' submenu, which gives access to most of the Commodore DOS commands. From the command line, 'maintain other' with a request for a display of all files beginning with "a" is achieved like this:

maintain o "$a*"	'Other' is abbreviated to its initial letter, and followed by the Commodore disk command within double quotation marks. The same technique can be used for scratching or copying files with the appropriate Commodore disk commands.
maintain o "s0:junk"	Scratches the file called "junk" from drive 0.
list	If you have a Superbase program in memory, 'list' shows the first 23 lines on the screen. Pressing return scrolls you through subsequent lines. If no program is in memory, Superbase gives the message "No program present"; so if you're not sure whether there's anything there, use 'list' to find out. When you've got output set to the printer (see below), 'list' prints out the program -- an important safety precaution.
list 100-	This variant lists all lines from line 100 onwards.
new	A companion command to 'list'. Use 'new' to clear out any program in memory -- not Superbase itself, but a Superbase program. You must do this before selecting 'execute' from the menu, if you want 'execute' to prompt for a program name.
quit	The command that does clear out Superbase itself. Pops you back into BASIC after closing any open files and generally tidying up the database. Use with care.
print	Whether used on its own, as here, or with a string or field name, as in the examples below, 'print' switches the direction of output to the printer. It remains in force until a 'display' comand is used. The results of an 'output' command, of 'list' or 'status'

	or 'directory' or 'catalog', will appear on the printer if one is connected. Note that the Superbase 'print' on its own does not cause a linefeed on the printer, unlike the equivalent BASIC instruction; to achieve that, you must enter 'print " "'.
print "Test"	In addition to switching output to the printer, this command actually prints the word "Test". Use this as a quick way of checking that the printer is set up correctly.
print [name]	This example prints data from the [name] field of the current record.
display	This command is the opposite of 'print', in that it switches the direction of output to the screen, until a 'print' command reverses it.
display @3,10"Screen 1"	Displays the message at column 3, row 10. Display co-ordinates may not exceed 80 for the column (40 where relevant) or 23 for the row (22 for Apple).
display [name][phone]	Shows the two fields on the screen one after the other.
find "listname"	An example of how to name an index key list. When you press return after typing the name, Superbase opens a new list with this name, and then calls up the usual record format with square bracket field markers for you to enter the search criteria.
tlen 23	Sets the text length to 23 lines. This allows you to print out help or memo screens on single pages. Use 'teen 60' to reset to a text length suitable for 11 inch paper.
plen 33	Sets the page length to 33 lines. This allows you to use normal length paper for half size pages.
rmarg 132	Sets the right margin on the printer to column 132. The right hand display margin is unaffected and remains at the width of the screen, 40 or 80.

None of the three commands above works for screen display. Only the next command, 'lmarg', has an effect and thus needs to be reset when you change from printed output back to screen output.

lmarg 30	Sets the left margin to column 30. Use 'lmarg' on its own to reset to the default value (i.e. the value set in the "start" program).
pdef 0	The first of the two printer set up commands, 'pdef' is concerned with the type of character and control codes that Superbase outputs to the printer. These can be either Commodore ASCII (values 0, 5, and 6) or standard ASCII (values 1 and 2). Values 1 and 5 are suitable for printers with Epson compatible control codes, and values 2 and 6 for printers with daisywheel type control codes, normally Diablo compatible. Value 0 suits Commodore compatible dot matrix printers.
pdev 0	The second of the two printer set up commands is here used as for a Centronics parallel interface connected to the computer's user port. See the Superbase Manual for details of the parameters which must be entered for other types of interface.

(The references for these last two commands apply only to Commodore versions of Superbase; different parameters and meanings exist for the Apple version, and are detailed in the appropriate manual.)

Use of the Command Line

A few tips to reduce frustration and improve efficiency.

Editing Commands

You can move the cursor along the line in either direction with the normal cursor movement keys. Likewise, the keys for inserting and deleting are available. The spaces between commands and the values that go with them are not compulsory -- you can just as well type 'lmarg20' as 'lmarg 20'. In fact, you can leave the line with any size gaps between the parts of the line, as long as there are no meaningless characters on it. Use the delete key or the space bar to remove junk from a line before pressing return, which you may do with the cursor at any position.

Quotation Marks

Double quotation marks are used to set off the name of a file or database, or a piece of literal text such as "Hello", or a string

for use with 'maintain o' (see above). You can omit the closing quotation mark provided there is nothing following the relevant name or text.

Colons and Multiple Commands

A great time saver. You can string any number of commands together on one line by separating them with colons. Instead of typing:

```
plen 66   <return>
tlen 60   <return>
lmarg 20   <return>
rmarg 75   <return>
print "Test   <return>
```

you can type:

```
plen 66:tlen 60:lmarg 20:rmarg 75:print "Test   <return>
```

You are only limited by the size of the command line area -- 159 characters or 79 characters depending on screen width. If your commands require more space, never mind: for this we have programs.

Recalling the Command Line

The best feature of the lot. By pressing the back arrow key (<tab> for Apple), you recall the last command line onto the command line area. You can now press return to execute it again, or make any changes you want and then press return. The line can be recalled as many times as you like, so you can afford to experiment freely, for example with output lines involving many field names, knowing that you do not need to retype the bulk of the line.

As the last sentence implies, command lines that are input following the selection of a menu option such as 'output' are retained for recall just like lines that originate by being typed in full. (In some early versions of Superbase you may need to supply the leading command word corresponding to the menu option itself before re-executing the line.) Here is a command line for setting up the printer which is designed to be repeated with different parameters for 'pdef' and 'pdev' until the correct result is obtained:

```
pdef0:pdev0:print "Test": <return>
```

Abbreviating Commands

Commands can be abbreviated in the Commodore systems. The principle is quite simple: the shortest unique form of the command longer than one letter is allowed. The final letter of the abbreviation must be in upper case. Here are abbreviated forms of some of the more commonly used commands:

```
database        daT
print           prI
display         diS
maintain        maI
directory       diR
select          seL
output          ouT
execute         eX
list            lI
prog            pR
```

Variables

Remember algebra? All that (a+b) and equation solving stuff. Variables are like that, but much easier. They work by "standing for" a value. The value can change -- hence the term "variable". The value of a variable changes either within a program, or because you assign a new value to it directly. For now, I'm concerned with how you set up and change variables directly, from the command line. Later we'll be using variables in programs extensively.

There are two kinds of variables in Superbase: string and numeric. They behave exactly like variables in BASIC programs, with one exception that I'll explain later. String variables are used to represent text characters such as:

```
Fred
Julie
salesfile
London
Pennsylvania Avenue
15%
(212) 456 6789
0098
```

The last three look like they should be numeric, but in fact they cannot be: "0098" would be just 98 as a number (all leading zeroes are removed), and the preceding two each have one or more non-numeric characters in them.

By convention, we refer in the text to the contents of a string variable within double quotation marks.

Numeric variables must be numbers. They cannot contain any characters other than 0123456789 and the decimal point. You can, however, give them a negative value. Some examples:

```
12
456
789.2356
-1
.3333
```

Naming Variables

There is a section in the Superbase Manual about this. Only the first two letters of a variable name serve to distinguish it from other variables. String variables must end with a $ sign; numeric variables must not. Avoid the various Superbase and BASIC command words.

You should study the Manual and the BASIC programming manual that comes with the computer to be sure you know what is an allowable name for each type of variable.

Setting Up Variables

This is done from the command line. You create a variable just by naming it for the first time as you assign a value to it. You may use calc if you wish:

```
calc x$="Henry James" <return>
```

Or you may use the BASIC command 'let':

```
let x=15 <return>
```

But there is no need. Simply type the name of the variable, an equals sign, and then the value you wish to assign to the variable:

```
x$="Henry":y$="James" <return>
```

A variable retains its value until a new value is assigned. Variables can have no value, and are then referred to as null. You give a string variable a null value like this:

```
x$="" <return>
or x$=chr$(0) <return>
```

The latter is better than the former, which can produce an error if used with the 'asc(x$)' function. Numeric variables are said to be null if they are zero.

If you want to clear out all the current variables, type 'clr' on the command line ('clear' for Apple).

Variables may be assigned the value of other values:

```
x$=y$:a=b <return>
```

You must not mix string and numeric variables. The following assignments are illegal and would generate an error message:

```
x$=b:a="Henry" <return>
```

But of course this is all right:

```
x$="000345" <return>
```

as the numbers are behaving like ordinary text characters.

Using Variables

By far the commonest use of variables is in programs, where they serve a number of purposes, as later examples will show. On the command line, variables are often used to simplify calculations. If you want to see the results of multiplying 250 by various numbers, you can set up a variable to hold 250 and then use 'display' to see the results of the different calculations.

```
a=250 <return>
display a*17.5 <return>  (shows 4375.00)
```

Recall the line with the back arrow key, then edit it to display a different calculation:

```
display a*23.65 <return>  (shows 5912.50)
```

The variable a could of course hold the result of a more complex step in a longer calculation. One thing to be aware of is the need to separate numeric variables with semicolons:

```
display a b <return>
```

will probably show 0.00, because Superbase treats the two variables as one -- 'ab'. But:

```
display a;b <return>
```

shows them separately.

There are many more uses of the command line and variables, which do not impinge on the routine operation of a Superbase database. As we progress through the remaining chapters, which go into programming in increasing detail, some of these uses will become apparent. Others you will discover for yourself once you have made the connexion between this powerful feature of the system and your specific requirements. These requirements can only be identified through careful analysis.

CHAPTER 6
PROGRAMS: THEORY, PRACTICE AND MANAGEMENT

The Need for a Plan

Without a plan, you cannot act really effectively. Perhaps when you become experienced with Superbase, you will be able to improvise programs on the spot to meet unusual needs, but as a relative beginner you are well advised to invest the extra time needed for analysis and planning. Even experienced programmers make careless mistakes, and one rarely sees a program more than five lines long that runs first time.

What do I mean by a plan in this context? Perhaps as a person with an interest in computers and software you have come across terms like flow-charts, systems analysis, or even structured programming. What I am proposing is not a complex and jargon ridden descent into confusion, but a simple exercise in logic, really little more than a formal statement of the kind of daily clerical activity that takes place in any office.

Some Sample Plans

Imagine you are standing in front of a large filing cabinet. You have a number of jobs to do, which involve searching through the files in the cabinet, making various lists of the contents, or changing some of the figures in the files. You don't do this sort of thing randomly, I hope. You work out what your objective is, and the best means of achieving it. Then you go to work, trying not to go mad with the boredom of such a repetitive task.

The outline plans I give below are intended to show how easily Superbase can cope with this sort of requirement, once you have transferred the data into a database -- usually the most tedious part of computerization. After each outline I give the Superbase instructions that would be needed to make the plan work, omitting details such as field names. Notice, though, that I have deliberately written the plans themselves in plain language, avoiding the use of Superbase terms.

The Debtors List

Everyone in business needs to know who owes how much. If you need to get cash in, the best way is to call up the people who owe you the most. This plan is the basis for a program that runs automatically in the few minutes it takes to get yourself organized near to a telephone.

1. Open the file of customer records.
2. Find all the records for customers who owe more than $500.
3. Make a list of their names, phone numbers and what they owe.

Superbase instructions: file, find, output.

A Re-ordering List

Instead of laboriously counting inventory, comparing it with the files, and making endless lists of code numbers and suppliers, you can let Superbase do the routine work while you do something more productive. Here is a plan that shows how a program can help you manage inventory levels more efficiently. It assumes that the inventory record card for each product you sell contains certain essential information, such as the minimum level of stock, details about the supplier, and whether the item is already on order.

1. Open the file of inventory records.
2. Find all the items with a quantity in stock below the minimum stock quantity, which are not already on order.
3. Make a list of the item code, description, price, supplier, and supplier's telephone number.
4. Mark the records to show that these products are now on order.

Superbase instructions: file, find, output, batch.

Updating a Running Total for Each Customer

Here everything depends on the nature of the records you keep. For this plan to work, it assumes that each customer record shows both a running total amount and any outstanding amount that has not been added to the running total. If there is nothing outstanding, this amount is set to zero. The program based on the plan would first of all make a list of all the records where there was an outstanding amount, and then use that as the basis for the rest of the processing.

1. Open the customer records file.
2. Read through all the customer records, and if you find an amount outstanding, add it to the running total for the customer and reduce the amount outstanding to zero.
3. Show the grand total for the outstanding amounts you found.

Superbase instructions: file, find, calc, batch, display, wait.

Dental Patients' Reminders

Provided the dentist keeps the computerized patient records up to date, there should be an improvement in dental health as the patients receive timely reminders to make an appointment. The plan also assumes that the dentist has had some standard letters pre-printed, to which the program adds just the patient's name and address and the date of the last visit.

1. Open the file of patients' records.
2. Find all the patients you haven't seen for six months, who had treatment for gum problems, and who you haven't written to within the last six weeks.

3. Type out reminder notes for them on the pre-printed half size notepaper, showing for each patient the name and address and the date of the last visit.
4. Update the file to show that you have now written to these patients.
5. Print out labels for the envelopes.

Superbase instructions: file, find, plen, tlen, output, batch.

Valuing a Stamp Collection

Just to show that I am aware that many Superbase users find the program a valuable adjunct to their hobbies, here is a philatelic application. The plan reasonably assumes that the record for each stamp shows which country it is from and the year of issue. The country information makes it possible to summarize by country.

1. Open the stamp collection file.
2. Find all the stamps in the collection worth more than $20.
3. Put them into order of country and date of issue.
4. Make a list showing the subtotal for each country.

Superbase instructions: report, find, sort, total, subtotal, detail, endreport.

The next two plans are slightly more complex, dealing with record data in ways that are somewhat closer to Superbase than to a manual system. I have used a few terms that you will recognize from Chapters 1 and 2.

Verifying Data Accuracy

It's important that a personal record is filled in accurately. Superbase can help to do this by imposing checks at the time when the record is first created. You may not be able to verify information about a person's address, but a program can easily eliminate contradictions between, say, date of birth and age.

1. Open the patient records file.
2. Fill in a record format with the details for the new patient.
3. Check that certain fields, such as age, date of birth, and sex, are within the ranges allowed for them.
4. If there are any errors, show what they are and insist on corrections.
5. When the record is correct, add it to the file.
6. Offer the operator options to add another record or quit.

Superbase instructions: file, clear, ask, select current, store, menu

Some BASIC instructions would also be needed if this plan were to be translated into an actual program.

Amending Record Data

Perhaps the situation demands that most of the information in a record is protected against accidental or unauthorized alteration, although a certain field has to be changed regularly by a filing clerk. For Superbase, it's easy to request information from the keyboard and transfer it to a single specified field, while the record itself is only displayed for confirmation, with no possibility of general editing allowed.

1. Open the customer file.
2. Call up the required record using the index key field.
3. Confirm that the displayed record is the right one.
4. Allow the appropriate field to be changed.
5. Confirm that the changed record is correct.
6. Store the changed record back in the file.
7. Offer the operator options to call up another record or quit.

Superbase instructions: file, select key, ask, select current, store, menu.

As in the previous example, some BASIC instructions would also be needed.

In this last example in particular, the steps of the plan seem to be giving instructions to Superbase to carry out various actions. This is exactly how you should go about programming. First, you decide what to do. Then you tell Superbase how to do it. Of course, you do have to learn the details of how to bend the system to your will, but you may have noticed two encouraging facts about the examples: the Superbase instructions include a number of the menu options which you already know, and certain instructions tend to be repeated. Because almost every user begins with the menu options and sets up several files before advancing to the programming stage, the amount of learning is reduced. The menu options are generally the bones of the program -- the stuff that has to be learned is the connective tissue. Also, programming tends to be repetitive, and Superbase is designed to take advantage of this with its powerful main options, which can handle a wide variety of different tasks. The result is that you soon become used to the Superbase vocabulary and able to use it effectively.

An Example: Programmed Record Deletion

This is the first worked example of programming in this book, and as such it has to be relatively straightforward, even compared to the simplicity of most Superbase programs. However, this is not a trivial example. Deleting records from Superbase files can be an extremely time consuming and frustrating task if you are restricted to the menu options alone. Any method of automating the process is sure to welcomed by many users of the system.

So where do we begin? As the previous chapter insisted, the prerequisite to any action is a plan of action. The prerequisite

to that is an understanding of the problem -- and a thorough understanding will reveal the solution.

Stating the Requirement

The problem is that from time to time you need to remove data from your database. Assuming that we are dealing with one file at a time, there are three possible courses of action:

1. Remove all the records from the file, but keep the empty file format on the database catalog (the list you see when you use 'file').
2. Remove all the records, the file format from the catalog, and the file definition from the disk -- total deletion.
3. Remove only a group of records from the file.

I shall develop each of these possibilities into a small program. For now, I'll skip the practical steps of how to write the program, except to say that you use the 'grog' option from Menu 2. Later in this chapter I deal with the mechanics of writing programs and storing them for future use.

Deleting All the Records

The objective is to remove all the records from the file, while leaving the file format on the database catalog for future use. Here is the plan:

1. Open the file; in the example, it's called "customers".
2. Remove the records automatically.
3. Leave the file format unaffected.
4. Return to the menu.

There are some menu options we can use in the construction of the program: 'file' opens the file for us, and 'select delete', from the 'select' submenu, is able to delete the current record. If we can use the latter repetitively, we should be able to create a short efficient program. So here's the first line of the program:

```
100 file "customers"
```

Every line of a program must have a line number. This is used internally, and it doesn't matter what interval you use between line numbers. It's a good idea to leave ample room for inserting new line numbers into the sequence. This first line uses the 'file' option from the menu, followed by the file name inside double quotation marks. Next, we select the first record in the file, just to position ourselves correctly for the deletion that comes next:

```
200 select f
```

Notice how 'first' is abbreviated to 'f': this is standard for the options on the 'select' submenu. Next, deletion of the current record, the one we've just selected:

```
300 select d
```

CAUTION. *You may be used to Superbase requesting confirmation before executing a record deletion when you choose 'delete' from the submenu; this is not done when 'select d' is used in a program.*

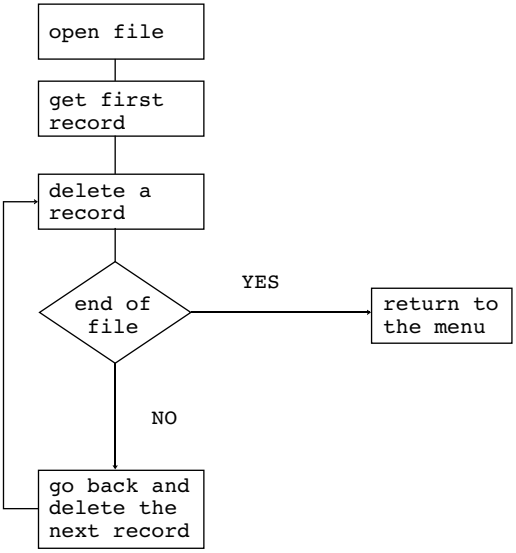
This disposes of the current record. Superbase now automatically selects the next record in the file, making it the new current record. Before we can go back to delete it, however, we must check to see that we have not reached the end of the file, as executing a record deletion at the end of the file has a fairly drastic consequence -- it removes the file format, which we specifically want to avoid. Here's the end of file check:

```
400 eof menu
```

The line uses the abbreviation 'eof' to test for the condition of end of file. Only if that condition is true -- it really is the end of the file -- does Superbase carry out any instructions to the right of 'eof' on the line. In this case we have simply the instruction 'menu', which directs the program to display Menu 1 and return control to the user. If the 'eof' condition is false -- no, it is not yet the end of the file -- Superbase carries on with the next and last instruction, which is to go back to the line that deletes the current record:

```
500 goto 300
```

Let's simplify this sequence with a flowchart diagram:



Flowcharts can be very useful for clarifying your thoughts when it comes to making decisions within the program, but too much emphasis on the technique can, in my opinion, interfere with the process of learning how the language itself works. One advantage of the flowchart is that you can see more clearly the meaning of the programming term "loop", which we use to indicate that an operation is performed more than once. Here the loop is from line 500 to line 300; lines 300, 400, and 500 will be executed repetitively until the end of the file is reached, when control returns to the menu. Here's the final version:

```
100 file "customers"
200 select f
300 select d
400 eof menu
500 goto 300
```

PROGRAMMER'S CHALLENGE. *What would happen if the file had no records in it to begin with?*

Before moving on to the next example, let's shorten the program by placing more than one command on a line where possible:

```
100 file "customers":select f
200 select d:eof menu
300 goto 200
```

Notice that we cannot use the line:

```
100 file "customers":select f:select d:eof
menu:goto 100
```

The instruction 'goto 100' would never be executed, as it occurs to the right of the 'eof' conditional.

Deleting All the Records and the File Definition

The objective is to remove all the records from the file, removing the file format from the database catalog, and the file definition from the disk. Here is the plan:

1. Open the file, for example, "customers".
2. Remove the records automatically.
3. Remove the file format.
4. Remove the disk file.
5. Return to the menu.

Basically, the program is the same as the last one, with two additions. This time I'll give the full listing and then explain it:

```
100 file "customers":select f
200 select d:eof 400
300 goto 200
400 select d
500 maintain o "s0:customers"
600 menu
```

The first line is the same: open the file and position to the first record. The second line deletes the current record and if the action detects the end of the file transfers control to line 400; if it's not 'end of file', line 300 transfers control back to line 200 to delete the next record.

Line 400 is the first addition. It deliberately executes one more 'select d' in the empty file. When this is done, Superbase removes the file from the database catalog; if you use 'file' or 'maintain c' the name will no longer be there.

Line 500 is the second addition. It uses the 'maintain other' option which I discussed in the previous chapter, to pass a command to the Commodore Disk Operating System (Apple users have a different set of disk commands) which instructs it to 'scratch' (abbreviated 's') the file called "customers" from drive 0 of the disk unit. This erases the file definition, meeting the requirements of step 4 of the plan. Finally, the 'menu' command transfers control back to the user at the main menu.

So, to remove a file completely, there are three steps:

1. Delete all the records.
2. Delete once more to remove the file name from the database catalog.
3. Scratch the file from the disk.

Deleting a Group of Records from a File

This is easier than the last job in that there is by definition no requirement to remove the file itself, either from the database catalog or the disk -- "a group of records" implies that there will be some remaining in the file.

Here is the plan:

1. Open the file, for example, "customers".
2. Remove the records from the specified list automatically.
3. Return to the menu.

For this program I assume that you have already used 'find' to create an index key list, called "h8list". As before, I'll give the full listing and then explain it:

```
100 file "customers"
200 select from "h8list":eol menu
300 select d: goto 200
```

The first line is almost the same; it opens the file but does not position to the first record. The reason for this is that we are using the instruction 'select from "listname"' to position precisely to each of the records whose index key appears in the list. If the file were to contain five records with index keys:

```
adams
baker
charles
david
edwards
```

and "h8list" contained just:

```
baker
edwards
```

then the first execution of 'select from "h8list"' would call up the record with the key "baker", and the second execution of the command would call up the record keyed by "edwards". So all the positioning, or calling up of current records, is done with this command. There is no need for 'select f' to call up the first record in the file.

Line 200 executes the 'select from' instruction. In the above example, what would happen on the third execution? Just as when we were deleting all the records in the file we had to use 'eof' to test for the end of the file each time a deletion was done, here we test for the end of the list with 'eol'. It works in exactly the same way, only allowing the commands to the right of itself to be executed if it is true that the end of the list has been reached. So, 'eol menu' controls the end of the program.

Line 300 contains the deletion instruction. It works as before, removing the current record and making the next record in the file

into the current record. However, this time there is no use for the next record, as the positioning is done with the separate 'select from', to which the last instruction in the program, on the same line, transfers control.

Managing Your Programs

This section brings together some of the common sense rules for the everyday use of Superbase programs. If you tried out any of the programs in the last chapter, you may already have discovered some of the rules for yourself. If you haven't yet written a program and can't see what the fuss is about, please take it on trust that unless you consciously manage your programs you can easily drift into a situation in which you have dozens of small routines spread across numerous different disks, and can never find the one you want. All the benefits gained from automating Superbase operations can be lost.

Writing a Program

1. Starting from Menu 1, type:

   ```
   new <return>
   ```

   This clears out any existing program from the program area inside Superbase.

2. Press return to select Menu 2, and then select option 5, 'prog'. This is the special Superbase program editor. When you select 'prog', you see a blank screen with the cursor positioned at the top left corner.

3. Type in the program lines, beginning each one with a line number. Enter the instructions you want in the order you want them, with due regard for the two elements of every program:

   Syntax. Every detail of each line must be correct.

   Logic. The program must follow the plan correctly.

   I give a number of editing tips at the end of the chapter, but here I must stress that you should always put the name, date, author and purpose of the program at the beginning, like this:

   ```
   1 rem Program "lstdel.p", Author BH, Date 12/12/85
   2 rem Purpose to delete records using a list
   3 rem Last modification 1/1/86
   ```

   The date of last modification is very important.

4. When you have finished writing the program, quit from 'prog' with 'fl STOP' or Control-Q. This returns you to Menu 1.

Storing a Program

Whether it's a new program or a modified old one, you must put a copy on the disk before you execute it. If there's an error in the logic, you could lose the carefully typed in program lines -- maybe hours of effort -- as the program goes into an uninterruptible loop, or inadvertently quits from Superbase (it doesn't happen often, but remember Murphy's law!).

On the command line, type:

```
save "name <return>
```

"Name" is the name you choose for your program. Superbase supplies a ".p" extension, so if you type "lstdel" and press return, Superbase will create a file on the disk called "lstdel.p".

The name you give should be unique, unless you want Superbase to overwrite an existing program of the same name. Sometimes this is useful, but if especially if you are experimenting with several versions of the same program, you must be careful to distinguish them. One method is the addition of a version number to the name:

```
lstdel.1.p
lstde1.2.p
```

But remember that the length of the program name is limited to 16 for Commodore systems and 30 for Apple (less for ProDOS), including the ".p" extension.

You can store programs on any valid drive by including the drive specifier. For Commodore systems this is done by typing the specifier before the program name. This does not form part of the name itself: 'save "0:lstdel" <return>'.

Executing a Program

When it comes to executing the program, you have four possible situations:

1. While still inside the 'prog' program editor, type:

   ```
   run <return>
   ```

   This immediately executes the program.

2. From Menu 1, if the program you want is already in Superbase's memory, select 'execute', option 7 on the menu. This immediately executes whatever is in memory. If there's nothing in memory (perhaps you typed 'new'), 'execute' prompts you to enter the name of the program you want. Type it in and press return. Superbase searches for this program on the disk.

3. To call up the program from the disk and execute it all in one go, type:

```
execute "name <return>
```

This automatically clears out any program already in memory.

4. You may want to look at or edit the program without running it automatically. In this case, use the 'load' command:

```
load "name <return>
```

Now, if you want to execute it, type 'run' or select the 'execute' option.

Looking at a Program

Sometimes it's convenient to look at the program in memory without using 'prog'. The quickest way to get the maximum number of lines on the screen is to type on the command line:

```
list <return>
```

If you are going to have several programs as part of your system, you must keep a folder with the listings of all the programs in it. This is a valuable extra safety measure, and it can help avoid duplicated effort. To obtain a printout of the program in memory, first set up the printer, then type on the command line:

```
print:list:display <return>
```

If you have wide paper, you may be able to set the right margin to 179, the maximum width of a Superbase program line in an 80 column system. Use 'rmarg'. You can also list a Superbase program to a disk file. I explain how in Chapter 11.

Your Program Library

It is vital that you keep track of all the programs in the system. As I said above, you must keep hard copies of every program listing. Here are some other suggestions to help you run a tight ship:

1. Label every disk clearly, with disk name and id (volume number for Apple). If possible, attach a list of what's on the disk to the disk sleeve.

2. Keep library disks that contain a spare copy of every program you write, in addition to the working backup disks.

3. Print out a directory listing for each disk. Do this by typing on the command line:

```
print:directory:display <return>
```

If you follow these simple rules your Superbase system stands a chance of remaining easy to manage.

Editing a Program

If you write a program, you'll have to change it. This is inevitable. No program of any size is bug free, certainly not when it is first used. So you need to be able to use the 'prog' editor fluently to make changes to program lines. As a preliminary, you need to load the program you want:

1. On the command line, type:

```
load "name <return>
```

This obtains the program you want from the disk.

2. Either select the 'prog' option from Menu 2, or type 'prog' on the command line. This enters the program editor, positioning the cursor at the beginning of the program. If you want to position at a specific line number, you can type:

```
prog 999 <return>
```

where 999 represents the number you want.

Editing Tips

You must learn the editor's facilities for yourself. They are all described in the first part of the Programming Section of the Superbase Manual. The keystrokes here refer to Commodore versions of Superbase. Apple users should substitute their equivalents.

1. To position quickly to the end of a program, first clear the screen with 'SHIFT/CLR', then press the cursor up arrow once.

2. Always press return on a line to register any editing changes.

3. Use 'shift / return' frequently to display the actual program code when editing, as your changes may obscure the real state of the line you're working on.

4. Learn and use the abbreviations for Superbase instructions.

5. Create duplicate lines quickly by typing the new line numbers over the original as many times as the number of duplicates you want, pressing return each time. Then use 'shift / return' to display the new lines.

6. To keep a program line available but non-executable -- so you won't have to type it in again -- insert 'rem' after the line number. Remove the 'rem' when you want the line to be active again.

7. When developing a program, make the last line to execute:

```
      999 wait:prog
   or 999 prog 100
```

   This causes the program to pause when it finishes until you press return, when it re-enters the program editor. The second version takes you to the indicated line number -- you can set it to take you to the part of the program you are working on.

8. Similarly, use 'wait' within the program at any time to make it pause until you hit return. More drastically, 'stop' as a program line causes the program to do just that -- stop dead. The Superbase 'stop', unlike the BASIC 'stop', does not allow you to continue execution from the next line. Use these as temporary aids to getting the program right, and remove them when it's finished.

9. You can manually renumber a whole group of lines starting at a high line number, say 9000, to temporarily remove them from the main code. Put a 'stop' immediately before them to stop them ever being executed. If you need them again, renumber them -- carefully -- to a new position in the program. ('Renumber' is a valid command only in Superbase 128.)

If you absorb and do your best to follow the advice I give in this chapter, you should find life with Superbase a lot easier. As you will start to see in the chapters that come next, programs can be constructed to handle just about every Superbase task, and once you get into the habit the programs used for quick solutions can increase in number dramatically, leading to a state of labyrinthine confusion.

CHAPTER 7
AUTOMATED SEARCH, SORT, UPDATE AND OUTPUT

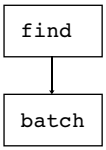
Part I dealt with the essentials of setting up a Superbase system. I covered the topics of file design, entering data, searching the file, sorting, updating and producing output. Now we are ready to string together some of these fundamental activities into automated sequences -- programs. But before we do, let's review the context in which the program is assumed to be operating.

Unless a certain minimum level of design is carried through, a Superbase system is unlikely to be capable of being programmed effectively. Files that lack search fields cannot be analysed into subgroups. If the need to sort is not anticipated, you are limited to output in the order of the index key only. It all comes back to the original design of the file, which must hold your data in fields that allow all the variations in output that you will need. Remember, the smaller the elements of the data, the more flexibility you have.

At the other end of the system, you must be clear about the details of the output you want. Do you want unstructured dumps of the record data, or tidy line by line tables? Perhaps you use pre-printed stationery, or you want to integrate Superbase's output with your word processor. Whatever the details of your requirements, you must think them through from beginning to end, from input to output, from file design to report layout. When this has been done, it becomes possible to create programs that take advantage of the system's internal consistency, making a collection of data into a responsive source of analysis and information.

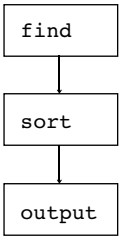
The Flow of Events

We are dealing mainly with the menu options of Superbase. There may be a need for other instructions in the programs we create, but at the centre of them are the key processing commands for selection, ordering, updating and output: 'find', 'sort', 'batch' and 'output' respectively. These can be be combined in a number of ways, such as:

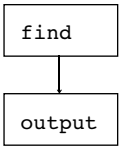


At its simplest, a Superbase program has just two steps. If it had one, it might just as well be a command line -- although there are

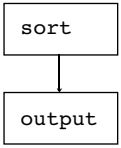
users who prefer to keep one line programs to save themselves the chore of typing field names. The diagram above, a simple flowchart, shows that you can have a program that first searches the file for a subgroup of records, then updates that group. With 'batch' there is no need for the data to be processed in other than index key order. Not so when output is required. Then sorting is frequently part of the process:



So we have what is essentially a three step program, although in many cases the sort is not required and it reverts to the simpler:



Of course there are times when a subgroup is not needed. Then it is 'find' that can be omitted:



In the next chapter we shall see that the number of steps increases significantly when the various instructions associated with the Superbase report generator are included, but for now we can contemplate with satisfaction the elegant power of a system that can achieve so much with so little. In BASIC, a program might well need fifty lines to achieve what Superbase does in three. And having contemplated, let's move on to the first of the core processing commands, 'find'.

Automated Searching: 'find'

The purpose of 'find' is to generate an index key list that identifies a subgroup of records in the main file. The list can then be used by a number of Superbase options to process the subgroup alone, leaving the rest of the file unaffected. In Part I, I looked at 'find' purely as a menu option, whereas here I shall be treating it as a program instruction. Although the syntax for 'find' used in this way is new to you, the logic of the various kinds of search criteria is the same. If you are not yet familiar with 'find', it might be advisable to check up on it before proceeding, as I shall not be repeating earlier explanations.

Naming the List

A 'find' program line has two parts: the basic instruction, and the search criteria. Here is an example of a simple line:

```
10 find "h8list" where [name] is "=Jones"
```

The list name, which is the word inside quotes right after 'find', can be any name, or simply the empty double quotes -- "" -- to refer to the system's default list, "h8list". The default list is just a term for the list that Superbase creates automatically if you don't name one. The blank record format as supplied by the menu option is here replaced by a single criterion in the form of a "where clause" -- a construction that specifies the value to be matched in a particular field. I'll be looking at where clauses in more detail in a moment.

You don't have to use the where clause. If it's omitted like this:

```
10 find "newlist"
```

Superbase calls up the usual blank record format, exactly as if the option had been called from the menu. The difference is that once the 'find' instruction has been completed, the program goes on to execute the next line automatically. It's a good idea to keep to a convention regarding list names. As well as making the list name reflect its function, you can use the word "list" or an abbreviation of it as part of the list name:

```
newlist
credit.lst
alist
blist
clist
list.may
list.jun
lst.dr.8602
```

If you have a dual drive system (not a dual unit system) you can save space on the database disk by creating the list on the other drive. Use a drive specifier:

```
1:list.1
0:list.2
```

If the naming convention uses a standard prefix such as "list" you can use it to gain a quick selective view of the lists on the disk with this command:

```
maI o"$0:list*"
```

You can add to an existing list (Superbase creates the list if it does not exist). The command is:

```
10 find "daylist,a"
```

This can be useful when you need to sort and print out say once a week but must update the file daily. Each day, you could use 'find' to tag the records to be printed out onto a single list, and 'batch' to update the file. The weekly sort puts the list into final order for printing. You would have to avoid deleting records from the file to be sure that the keys in the final sorted list were still all valid.

The ",a" extension to the list name could also be used for "picking" individual records: adding specific index keys to a list as you browse through a file with 'select', although the file would have to be pretty short for this to be workable. Here's the procedure:

```
10 find "picklist,a" where [key] is "←"
```

With this one line program in memory, you browse through the file with 'select'. At any time, you can return to Menu 1 and press f7 for 'execute'. Superbase takes the back arrow as an instruction to request the search criterion on the command line. You type in the exact key or keys (as in "=Smith/=Jones/=Brown" etc. -- don't forget the equals signs) that you want to add to the list and press return. Superbase searches the whole file -- hence the need for a short one -- adds any keys it finds to the list and returns to the menu. You can then resume browsing. There is a more efficient way of achieving this end that is acceptable for any size file, which I discuss in Chapter 10.

The 'where' Clause

As I indicated above, this is the programmed equivalent of the blank screen into which you type the search criteria. The 'where' clause specifies a field name followed by the word 'is' and either a criterion within quotation marks or a string variable. I'll leave the use of variables until the next section.

You can have multiple criteria after the 'where', separated by semicolons. You may not repeat a field name. The maximum number of criteria for each 'find' depends on the length of the field names, the length of the criteria, and whether you have a 40 or 80 column

screen. If you can't fit all the criteria you want into a program line, you have to call up the blank screen and enter them manually; this is done by restricting the line to:

```
10 find "alist"
```

Using this technique the maximum number of search criteria is practically unlimited.

The formats of the various types of criteria are essentially the same as for the 'select match' command discussed in a previous chapter. This is true for text and numeric searches, pattern matching, sliding searches, and both range and alternative based searches. Here are the programmed equivalents of the search criteria discussed in Chapter 4. First, exact matches in two fields:

```
10 find "" where [name] is "=John Smith"; [city] is
"=London"
```

A sliding match only, within the [name] field:

```
10 find "" where [name] is "Smith"
```

Pattern matching search combined with sliding match in the [street] field only:

```
10 find "" where [street] is "? second ?"
```

Exact search in [code] field for all entries beginning with "c", plus exact search in [zip] field for all entries beginning with "069":

```
10 find "" where [code] is "=c*"; [zip] is "=069*"
```

The opposite, specifically excluding entries beginning with "c":

```
10 find "" where [code] is "#c*"; [zip] is "=069*"
```

Search across multiple [comment] fields terminated in [status] field:

```
10 find "" where [comment-1] is "whales-"; [status]
is "*"
```

Search for [code] entries beginning with "a", "b" or "c", plus any one of the three cities specified:

```
10 find "" where [code] is "=a*/=b*/=c*"; [city] is
"London/Paris/Rome"
```

Search for a range within the [age] field:

```
10 find "" where [age] is ">30&<50"
```

The same, but including the 30 and 50 year olds:

```
10 find "" where [age] is ">29&<51"
```

The same, but excluding the 30 to 50 year olds:

```
10 find "" where [age] is "<30/>50"
```

Searching for dates in a particular month:

```
10 find "" where [date] is ">30Sep85&<01Nov85"
```

Note that although this finds all dates in October 1985, the month itself is not referred to. Dates are a special case, all the more important for being so often required during 'find'. This example shows how to construct a 'where' clause for a range of dates, which is useful, but not as useful as a program that requests new date information every time the program is executed.

A way of inputting criteria for a search each time a 'find' program runs is by means of the back arrow delayed request character, which I referred to earlier in this chapter. If you put this character in the 'where' clause instead of a specific criterion, Superbase will request input for the specified field (or fields) on the command line. This is easy, but there are editing tricks involved. Since the back arrow key has a special function within the 'prog' program editor you have to enter it in one of two ways. Either you type the program line on the command line, or, in the program editor, you assign the ASCII code value of the character to a string variable and use that in the 'where' clause.

If you use the command line approach, tap the space bar first to call up the command line area, then type the line number (being careful not to use the line number of an existing line), then the 'find' line in full, enclosing the back arrow character in quotation marks, as in this example:

```
<space> 10 find "" where [name] is "←" <return>
```

Pressing return enters the numbered line into the program area. You can now enter 'prog' and edit the line further if need be. If you want to set this up entirely within 'prog', you must use a string variable:

```
10 a$=chr$(95): rem 95 is ascii for back arrow
20 find "" where [name] is a$: [city] is a$: [zip] is a$
```

Notice that a$ can be repeated as often as you wish. Unfortunately, this approach does not work with dates, which have to be handled using the 'ask' statement; this is discussed in Chapter 9.

Automated Updating: 'batch'

The 'batch' command is used either to update the file or part of a file, or to perform fairly simple calculations on the file using variables set up for the purpose. Often, 'batch' is used following a 'find' command. 'Find' identifies a group within the file that needs updating, and then 'batch', like an automatic editor, goes through the list and makes the necessary changes. At its simplest, a program to do this would have two lines:

```
10 find "orderlist" where [amount] is ">0"
20 batch from "orderlist" [total]=[total]+[amount];
[amount]=0
```

The program makes a list of all records where the outstanding amount is above zero, then, for each of the records on the list, adds the amount to the [total] field and resets the [outstanding] field to zero. The example obviously refers to numeric fields only. We could of course update text fields just as easily, placing the new value for the field within quotation marks:

```
10 find "checklist" where [status] is "not done"
20 batch from "checklist" [status]="ok"
```

Notice in the first example that 'batch' allows multiple update clauses, separated by semicolons, rather like 'find'. In 'batch', you may not modify a field with a field that has already been modified in the same line. This would be illegal:

```
20 batch from "" [status]="ok";[status-a]=[status]
```

The reason for the rule is that Superbase updates the record according to the values in its fields when it is first read from the disk.

Using 'batch' to Calculate Totals

Now let's turn to the first serious use of variables in this exploration of Superbase: for computing totals. 'Batch' has two main functions: reading records and updating them. We can make use of the record reading function to perform certain useful jobs, such as counting the number of records in a list, adding up totals for any of the fields in a record, finding averages from the two previous operations, finding maximum and minimum values for fields, and providing displayable results. To do this we need to precede the 'batch' operation with some program lines that set up the numeric variables we want. Although Superbase in fact initializes every new numeric variable to zero, it is a rule of programming that the variables are defined and initialized at the start of the program. Here's the simplest example, a program that counts the records in a list:

```
10 t=0
20 batch from "" t=t+1
30 display t
40 wait
```

As 'batch' reads each record on the list, it adds one to the previous value of t, which acts as a tally. If t were not set to zero in line 10, the initial value could be wrong, in which case the error would be transmitted through to the end of the operation. It's important to notice the final two lines here; we'll be using them a lot. If you don't instruct Superbase to show its results and then pause, it will just go back to the menu. The 'wait' instruction ensures that the user must press return to terminate the program, and the result of the preceding 'display' thus remains on the screen until a key is pressed.

It is also possible to have a running display of the variable t as the batch operation proceeds. This is achieved by placing the variable on its own at the end of the program line:

```
20 batch from "" t=t+1;@0@4,4"Running total is:"t
```

As the example shows, the statement will also accept positioned text items. The next example shows how to add up totals for two fields, and display the results:

```
10 t1=0:t2=0
20 batch from "" t1=t1+[amount];t2=t2+[total]
30 display "Total for amount field:"t1
40 display "Total for total field: "t2
50 wait
```

Now the two types of operation can be combined to produce a count, totals, and averages:

```
10 t=0:t1=0:t2=0
20 batch from "" t=t+1;t1=t1+[amount];t2=t2+[total]
30 display "Total for amount field:"t1;"Average:"t1/t
40 display "Total for total field: "t2;"Average:"t2/t
50 display "Count of records in list:"t
60 wait
```

There is no reason at all why the counting operations should not be included in the line we used earlier to update the file. It would look like this:

```
20 batch from "" t=t+1; t1=t1+[amount]; t2=t2+[total]
+[amount]; [total]=[total]+[amount]; [amount]=0
```

We are careful to increment (jargon for add to) the total variable t1 for [amount] before resetting [amount] to zero. However, to obtain a valid grand total, we must increment t2 with both [total] and [amount], as we are not allowed to refer to [total] after [amount] has been added to it. A trifle complex, but it does a lot of work.

Maximum and Minimum Field Values

The next example is somewhat more difficult to grasp. We can use 'batch' to work out maximum or minimum values for any given field, say [amount]. This is done by evaluating a Boolean expression such as ([amount]<mx) each time a record is read, and using the result to determine whether the [amount] field in the current record counts as the new maximum or minimum figure. At the end, a variable contains the highest or lowest figure encountered during the operation. The sequence of operations is crucial; here is the code for obtaining a maximum value:

```
10 x=0: rem x will evaluate to 0 or -1 for each record
20 max=0: rem max is incremented when [amount] is higher
30 batch all x=([amount]<max)+1; max=max+([amount]*x)
-(max*x)
40 display "Maximum value of amount field is:"max
50 wait
```

Here's how it works. The first expression, 'x=([amount]<max)+1', evaluates to 0 if the [amount] field in the record is less than the current value of max, or 1 if [amount] is greater than max. The second expression, 'max=max+([amount]*x)-(max*x)', leaves max as it was if x is 0, but if x is 1 it first adds the new [amount] to max and then subtracts the old value of max, leaving max equal to the new [amount].

The operation to obtain a minimum value is almost exactly the same, but we must initialize the variable min to its highest possible value instead of zero:

```
10 x=0: rem x will evaluate to 0 or -1 for each record
20 min=999999: rem min decrements when [amount] is lower
30 batch all x=([amount]>min)+1; min=min+([amount]*x)
-(min*x)
40 display "Minimum value of amount field is:"min
50 wait
```

The '<' sign in the first expression becomes '>', but otherwise there is no logical change in the line.

Automated Sorting: 'sort'

Sorting is always an intermediate step in a series of Superbase operations, never an end in itself. The end result of sorting a database file is a list of the index keys to the records in the file, arranged in the order specified in the 'sort' instruction. The database file itself remains unchanged, both in index key content and order. The list you end up with can be used in any command that uses lists, but it only makes sense with 'output', 'detail', which is one of the 'report' group of commands, or the programmed equivalents of these, a sequence of selection and print or display statements.

The objective is always to present the records from the file in a

different order from the basic index key order. You might want to store a file of customer records in the basic order of name, but print them out in the order of the cities where they live. In this case you would sort on the city field before printing:

```
10 sort all on [city] to "sortlist"
```

There are several examples of 'sort' commands in Chapter 4, illustrating the syntax for sorting on more than one field, using a list instead of the whole file, using the default name "h8list" as the name of the sorted list, and for achieving a descending sort.

Many of Superbase's commands let you use BASIC expressions in combination with field names from the record format. For example, this kind of display command is perfectly valid:

```
display left$([name],3) <return>
```

Provided there is a currently selected record, the first three letters of the [name] field will be displayed. This is useful, particularly in more advanced programming. However, no such expressions are allowed with 'sort'. A line like this is illegal:

```
10 sort all on mid$([name],3,3) to "sortlist"
```

The only variable factor is the length of the basis for comparison, which is specified with the truncation character '&', as in this line:

```
10 sort all on &15[name] to "sortlist"
```

A Digression: How 'sort' Works

The above example would use the first 15 characters of [name] as a basis for comparison, instead of the default 10. Sorting works by assembling in memory a string made up of the comparison strings from all the specified fields, followed by the record key. This sort string is then ordered with the sort strings from succeeding records until the memory is full, when Superbase writes a special file, "hslist". "Hslist" contains index keys only if the 'sort' was accomplished in just one phase, but if the memory space for sorting is filled up on the first phase, and another phase is needed, Superbase temporarily stores the full sort string for each record in "hslist". Since the maximum length of the sort string is 254 characters, "hslist" can become very large, so if you are planning a big 'sort', involving many fields and a large file, ensure that there is sufficient room on the disk. After the final phase, Superbase merges the keys from the sort strings in memory with the keys from "hslist", eventually producing the destination list that was specified in the 'sort' statement.

Record Formats Designed for Sorting

The requirement to use only simple field names, as in the previous example, and to avoid BASIC string functions rules out a lot of clever manipulation of record order based on the index key, which as you will recall can easily be designed with several component parts, each having a special meaning. But all is not lost. If you design your record format carefully, you can set up fields specially for use with sort operations. Here's an example:

```
code      <citi.aa.0123    >

sort-1    <aa       >

sort-2    <0123     >
```

"Citi" is an alphabetic bank code, "aa" is a type code that shows what kind of transaction this is, and "0123" denotes the day and the month. The two fields, [sort-1] and [sort-2] exist purely for use during sorting. They reproduce information from the main index key, and allow the file to be sorted in several useful ways, such as:

by type code
by month and day
by month and day within type code
by type code within month and day

For aesthetic reasons, such fields are normally tucked away on a screen by themselves, away from the main record data.

Perhaps you are thinking, "Surely this doesn't mean I have to type that information in all over again?" You don't. Remember 'batch'? We can use it to copy the data from the [code] field into the [sort-1] and [sort-2] fields -- automatically. Let's suppose you have just entered in some records with index keys of the kind shown above. Your program to find the new records, update them, and then sort the file by month and day within type code would look like this:

```
5  a$=chr$(0): rem To detect empty fields
10 find "" where [sort-1] is "="+a$
20 batch from "" [sort-1]=mid$([code],6,2);
[sort-2]=right$([code],4)
30 sort all on [sort-1][sort-2] to "sortlist"
```

This uses the BASIC string handling functions 'mid$' and 'right$', which allow you to refer to portions of a variable, or, as in this example, a Superbase field name. Check the explanations in your BASIC manual, as you should be familiar with them and other similar functions before attempting to do much programming. An essential prerequisite of using such functions in this way is that

they always refer to the same portion of the field. To ensure this, you must take care that the original keys are all of the same length.

What happens if the contents of a sort field are the same in another record, or in several records? If there is another sort field in the line, it determines the sorted order. So, if we had two records with the same type code -- [sort-1] -- but different day/month codes -- [sort-2] -- the records would be sorted by [sort-2]:

```
aa        0123
aa        0124
ab        0123
ac        0123
ac        0127
```

But what if the contents of the final sort field in the line are the same in more than one record? In this case, the index key itself determines the order. If you have specified unique keys for the file, all will be well. But if you have used duplicate keys, Superbase will not be able to produce satisfactory results, losing its way either when reading the index keys from a list made with 'find', or when you go to output using the sorted list. So, as I've advised before, avoid duplicate keys.

The idea of special fields can be put to other uses. I have explained earlier that Superbase allows either an ascending or a descending sort, but not both. However, there are times when you may want to combine the two. For instance, following through the current example, you may want to produce a list by month and day with the transactions for each day listed in order of magnitude, from largest to smallest. This requires a sort by [sort-2] in normal ascending order, with the amount on each record sorted in descending order. To achieve this, you need to set up a result field that subtracts the actual amount from a number larger than the largest allowed in the numeric field format. Here is the extended record format, with the numeric format and result field calculation shown to the right of the relevant fields:

```
code      <citi.aa.0123    >

sort-1    <aa       >

sort-2    <0123     >

amount    <   789.50>    format: ££££.££

sort-3    <  9210.50>    format: ££££.££
                      calculation: 10000-[amount]
```

To illustrate this further, let's consider a series of amounts and their corresponding complements as calculated by [sort-3]:

[amount]	*[sort-3]*
12.50	9987.50
456.23	9543.77
999.95	9000.05
1008.00	8992.00
5887.01	4112.99
9995.00	5.00

Now, when the file is sorted by [sort-3] in normal ascending order, the order of magnitudes in [amount] is reversed. The largest [amount], 9995.00, comes first, because its corresponding [sort-3], 5.00, is the smallest. The line to sort using the month/day code and the special calculation looks like this:

```
10 sort all on [sort-1][sort-3] to "sortlist"
```

Later, when you go to output using the sorted list, the printout will show the month/day codes in ascending order and the amounts in descending order:

```
0123          9995.00
0123          5887.01
0123           456.23
0126          7568.50
0126           856.00
0126            45.00
0127          3002.00
0127           563.12
0202          9112.00
0202           750.00
```

You can use the same technique with date fields, but the result field formula should be based on the largest possible Superbase date: 31 December, 1999. This has the numeric value 36525, so a formula that produced a complement to a field called [date] would be:

```
36525-[date]
```

Finally, a slightly more advanced trick for obtaining a similar capability with text fields. The objective is to generate a sort field that will allow a reversal like this:

Input data for [textfield]	*Print order*
10A	4B
32G	10A
4B	32G
621C	621C

'Batch' is useful, but we also need a string variable set to zeroes:

```
10 a$="000000":rem Enough zeroes to cope
20 batch all x=len([textfield]);[sort-4]=
left$(a$,6-x)+[textfield]
```

The contents of [sort-4] will be:

```
00004B
00010A
00032G
00621C
```

This corresponds to the desired print order shown above, so all you have to do is sort on [sort-4] and output using the resulting list.

Automated output: 'output'

So far this chapter has covered the major processing activities of searching the database, updating records, and sorting. Now we look at the end product of most systems: output. This can be automated too, so that complex and lengthy commands can be stored in permanent form and recalled with just a few keystrokes.

Let's recapitulate the most important features of 'output', which I explained in a previous chapter:

1. 'Output' is a repetitive command: it processes either the whole file or the records whose keys appear in a list named in the command line, outputting specified details record by record.

2. There are three kinds of output: screen, printer, and disk file. You select screen output with 'display', printer with 'print', and disk file with the variant syntax 'output ... to "filename"'. Whichever of 'display' or 'print' was last used remains in force until its converse is used.

3. You can include different kinds of item in your 'output' line: fields, text, formulas and BASIC variables, expressions and functions.

4. You can make the output items appear either 'across' the paper, line by line, or 'down', page by page with one item per line. These commands are alternates, like 'print' and 'display'.

5. Positioning commands allow you to set the row and column for any item.

6. Formatting commands allow you to set the shape of numeric items, remove trailing spaces, output blank lines, or cut fields short to a specified number of characters.

7. There are a number of commands to determine such things as margins, pause at end of page, and length of paper.

The procedure for automating the 'output' line is basically very simple. You just write a normal program line with the 'output' syntax following it:

```
10 output all [name][street][city]"Phone:"[telephone]
```

Methods of formatting and positioning have already been explained, together with the differences between line by line and page by page style output. Before going on to some more advanced ways of dealing with output in programs, I'll show how the examples from the previous chapter translate into program lines. The straightforward unformatted line by line style of output is achieved with lines like the one above. A more complex line is this:

```
10 output all @5[name] @25&5,2[balance] @40[telephone]
@5[street] @5[city] @5&[state] [zip]
```

Which produces:

```
John Jones          678.89   503 456 7891
21 Main Street
Winesburg
OH 34567
```

A similar line omitting the [balance] and [telephone] fields would be suitable for single column labels.

Page by page output could be obtained as follows:

```
10 output all down @5,1[name] @55,60&5,2[balance]
```

Two examples of disk file output for word processing are, first, a file with commas between the fields:

```
10 output all across to "datafile" &[name]","
&[street]","&[city]","&[state]&[zip]
```

Second, a file in a format suitable for Easy Script or Superscript:

```
10 output all fill to "datafile" [name][street][city]
[state][zip]
```

Programming for Output

Once you have decided to use Superbase's programming feature for your output, you gain certain advantages:

* You can separate out the statements for 'print', 'display', 'across' and 'down'.

* You can separate the statements that change the parameters of the output, such as page length or continuous print flag.

* You can print (but not display) column headings before starting the output.

* You can set up variables for use in the 'output' line.

Usually it's best to keep all the statements that control the output together, near the beginning of the program:

```
10 print:across:plen 33:tlen 30:cont 1
```

This line switches on printer output, line by line, with a paper length of 33 lines allowing 30 lines of printing (i.e. a three line bottom margin). Continuous printing is also set.

Likewise, if you will be needing variables, set them up at the beginning of the program:

```
10 s$=" ":c$=",":crS=chr$(13)
```

Here, s$ will be used to print a space, c$ a comma, and cr$ a carriage return. In the following example, cr$ ensures a blank line before each record.

The other advantage is the ability to do direct printing before you begin the main output. In this example, 'print' is used to produce a main heading and column headings:

```
20 print @22"CUSTOMER LIST"@1@5,0"Name and address"
@28"Balance"@39"Telephone"
30 output all cr$@5[name] @26&5,2[balance] @39[telephone]
@5[street] @5[city] @5&[state] [zip]
```

Notice that although the column positioning parameters 5 and 39 are the same for both the column headings and the elements in the 'output' line, the parameter for "Balance" is different; this is because is better to align the heading of a numeric column with the right end of its field. You can easily incorporate simple graphic effects by printing lines of dashes at appropriate places:

```
20 print @22"CUSTOMER LIST"@1@5,0"Name and address"
@28"Balance"@39"Telephone"
25 print @5"----------------------------------------"
30 output all cr$@5[name] @26&5,2[balance] @39[telephone]
@5[street] @5[city] @5&[state] [zip]
```

If you had the foresight to assign the dashes to a variable you could use it in more than one place:

```
15 d$="----------------------------------------"
20 print @22"CUSTOMER LIST"@1@5,0"Name and address"
@28"Balance"@39"Telephone"@5d$
25 output all @5[name] @26&5,2[balance] @39[telephone]
@5[street] @5[city] @5&[state] [zip]@5d$
```

Your output would look like this:

```
                CUSTOMER LIST

Name and address    Balance     Telephone
-------------------------------------------
John Jones          678.89      503 456 7891
21 Main Street
Winesburg
OH 34567
-------------------------------------------
Mary Valdez         456.99      912 234 4564
1212 Pine
Laguna Beach
CA 93456
-------------------------------------------
```

A great improvement for a small effort.

Labels

Before we move on, a word about label printing. Superbase comes with a special Superbase program for labels, which is described in help screens and/or the Manual, depending on your version. This program is really intended for use when you require multiple columns of labels. Single columns of labels are best printed with a simple 'output' program. You just print all the name and address details at the same column position, separating one label from the next by printing the necessary number of carriage returns:

```
10 cr$=chr$(13):rem Set cr$ as carriage return
20 output all @5[name]@5[street]@5[city]
@5&[state][zip]cr$cr$cr$cr$cr$
```

We have four lines of name and address ([state] and [zip] being on the same line), and I'm assuming that the number of lines from the top of one label to the top of the next is nine; that means we must print five carriage returns to keep the printout regular.

Extending the Output Line: 'plus'

'Output' needs a small program if it is to be used most effectively. The reason for this is that when 'output' is selected as a menu option the number of field names that you can type in is limited by the length of the command area: either 79 or 159 characters, depending on screen column mode. The maximum for a 79 character line is 25, assuming you have single character field names. Since Superbase allows up to 127 fields per record, 'output' in menu mode can only be used to about 25% capacity. Even the longer line allows no more than one third capacity. The solution lies in the secondary Superbase statement, 'plus'. This provides a way of extending an 'output' program line over as many lines as you need.

Extended lines are less likely when you are printing line by line, as only a few fields can often exceed the usual 80 columns of a

printed line. But when a wider printer is in use, or the output is to be printed page by page, the need for several lines of output fields is not uncommon.

First, let's take an earlier example and reshape the line purely to illustrate the use of 'plus':

```
20 output all @5[name] @25&5,2[balance] plus
30 @37[telephone] @5[street] plus
40 @5[city] @5&[state] [zip] plus
50 @5d$
```

As you can see, 'plus' is typed as the last word on each line that is to be followed by an extension of the main line. The main 'output' statement itself is not repeated, and the final line must not have 'plus' at the end. In effect, all the lines from 20 to 50 are one line.

('Plus' can be used with some other statements: these are listed in the Manual in the Programming section, part 2. You cannot use 'plus' with 'find', 'batch', or 'sort'.)

Now let's consider an example where 'plus' would be essential to obtain the desired result. Suppose we are using a file format set up like an invoice, holding not only the customer's name and address but the invoice information as well. Such a file might look like this:

```
Invoice number      <         >
Invoice date        <         >

Customer:
Name      <                   >
Street    <                   >
City      <                   >
State     <   >  Zip  <       >

Quantity    Description       Price            Total
q1 <     >  d1 <          >  p1 <         >   t1 <         >
q2 <     >  d2 <          >  p2 <         >   t2 <         >
q3 <     >  d3 <          >  p3 <         >   t3 <         >
q4 <     >  d4 <          >  p4 <         >   t4 <         >
q5 <     >  d5 <          >  p5 <         >   t5 <         >
q6 <     >  d6 <          >  p6 <         >   t6 <         >
                                  Subtotal      <         >
                                  Tax           <         >
                                  Total         <         >
```

This is a total of 34 fields, so 'plus' is definitely needed. Now imagine a typical invoice form. The customer's name and address appears at top left, and below it on the right is the invoice number and the date. Then the lines of the invoice appear,

followed by the tax and total information. An 'output' program to print the invoice details at the correct points on the invoice form could look like this:

```
10 print:across
20 output all @1,1[name]@1,2[street]@1,3[city]plus
30 @1,4&[state][zip]@45,5[number]@45,6[date]plus
40 @1,9&5,0[q1] @15[d1] @40[p1] @55[t1] plus
50 @1,10&5,0[q2] @15[d2] @40[p2] @55[t2] plus
60 @1,11&5,0[q3] @15[d3] @40[p3] @55[t3] plus
70 @1,12&5,0[q4] @15[d4] @40[p4] @55[t4] plus
80 @1,13&5,0[q5] @15[d5] @40[p5] @55[t5] plus
90 @1,14&5,0[q6] @15[d6] @40[p6] @55[t6] plus
100 @55,16[subtotal]@55,17[tax]@55,18[total]
```

The program would produce this kind of output:

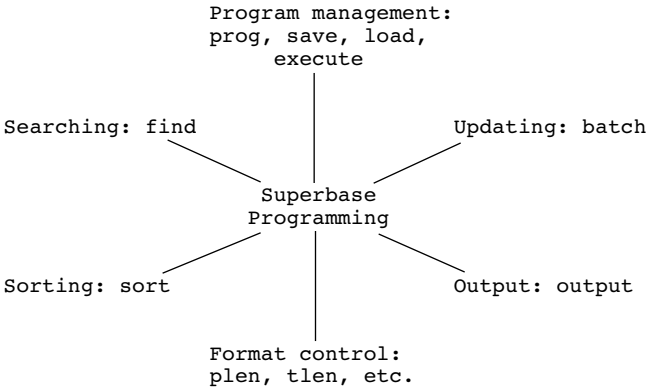
```
Mary Valdez
1212 Pine
Laguna Beach
CA 93456

                                00123

                                Jan1386


2          Plant holders       16.99      33.98
3          Seedling trays       3.45      10.35
                                           0.00
                                           0.00
                                           0.00
                                           0.00


                                          45.33
                                           3.17
                                          48.50
```

The program prints each invoice record on a new form, starting each one at column 1, row 1 -- the position of the [name] field. Notice how on lines 40 to 90 the row specifier is included only once, with the first item on the line. The other items are positioned on the same line, until a new row specifier is encountered.

Automated Processing: A Summary

In the next chapter, I'll be looking at the group of statements associated with Superbase's 'report' function. But before we move on, a summary of the kinds of programmed operations we've covered in this chapter will help to drive home the message that life with

Superbase can be made ridiculously simple. This diagram represents the key activities as aspects of a single function: Superbase programming.



All you, the user, have to do is decide which are your standard operations, and then for each one work out the sequence of steps, code it, and store it on disk. Then a simple command of the form:

```
execute "programname" <return>
```

will run the entire operation while you either sit back and watch or get on with something more congenial or productive.

Finally, a sample program that combines many of the operations we have looked at separately in this chapter. This invoice processing program is designed to do a number of jobs:

1. Search the invoices file for all invoices for the month of December 1985 with an outstanding balance.

2. Sort them by customer, date and order of magnitude.

3. Print out a credit control report detailing invoice number, customer name, date, invoice total, current balance, and customer's telephone number.

4. Ask the operator to insert the invoice stationery in the printer.

5. Search the file for all invoices with a status of "new".

6. Print out the new invoices.

7. Update the new invoices with a status of "printed".

8. Ask the operator to insert the normal stationery.

9. Print a summary list of the new invoices, with a figure for the total billed.

10. Ask the operator to insert the roll of labels in the printer.

11. Print out labels for the invoices.

12. Ask the operator to insert the normal stationery.

13. Reset format parameters, display a "Finished" message, and return to the menu.

The program makes some important assumptions about the invoice record format, which you should know before trying to understand how it works. The record format is basically like the one shown above, but with some extra fields:

[balance] Set to the same value as [total] when the invoice is first raised, but is then reduced as payments come in. If it is not zero, the balance is outstanding.

[telephone] Holds the customer's telephone number.

[status] A constant field set to "new" when the invoice is first raised, later updated to "printed".

[sort-1] A field that computes a value to allow the invoice amount to be sorted in descending order while the customer and date fields are sorted in ascending order.

```
10 rem Program name "invoice-1"
12 rem
14 rem *** SET UP INITIAL VALUES ***
16 rem
20 cl$=chr$(147):rem To clear the screen
24 cr$=chr$(13):rem To be used for label printing
26 pr$="Press return when ready":rem Used often
28 t=0:rem Initialize total for 'batch' count
30 file "invoices
42 rem
44 rem *** FIND AND SORT OVERDUE INVOICES ***
46 rem
50 find "inv.list" where [balance] is ">0";[status] is
"=printed"; [date] is ">30NOV85&<01JAN86
60 sort from "inv.list" on [name][date][sort-1]
to "inv.sort"
72 rem
74 rem *** SET PARAMETERS, PRINT CREDIT CONTROL LIST ***
76 rem
80 print:across:space 1:cont 1
```

```
90 output from "inv.sort" [number][name][date]&5,2[total]
&5,2[balance][telephone]
92 rem
94 rem *** CHANGE STATIONERY, RESET PARAMETERS ***
96 rem
100 display cl$"Put invoice stationery in printer"
@1,3pr$:wait
110 space 0
112 rem
114 rem *** FIND AND PRINT NEW INVOICES ***
116 rem
120 find "new.inv.list" where [status] is "new"
130 print
140 output from "new.inv.list" all @1,1[name]
@1,2[street]@1,3[city]plus
145 @1,4&[state][zip]@45,5[number]@45,6[date]plus
150 @1,9&5,0[q1] @15[d1] @40[p1] @55[t1] plus
160 @1,10&5,0[q2] @15[d2] @40[p2] @55[t2] plus
170 @1,11&5,0[q3] @15[d3] @40[p3] @55[t3] plus
180 @1,12&5,0[q4] @15[d4] @40[p4] @55[t4] plus
190 @1,13&5,0[q5] @15[d5] @4O[p5] @55[t5] plus
200 @1,14&5,0[q6] @15[d6] @4O[p6] @55[t6] plus
210 @55,16[subtotal]@55,17[tax]@55,18[total]
222 rem
224 rem *** UPDATE NEW INVOICES, GET TOTAL ***
226 rem
230 batch from "new.inv.list" [status]="printed";
t=t+[balance]
232 rem
234 rem *** CHANGE, RESET, PRINT SUMMARY ***
236 rem
240 display cl$"Put normal stationery in printer"
@1,3pr$:wait
260 print:space 1
270 output from "new.inv.list" [number][name][balance]
278 rem Total in next line comes from 230
280 print cr$"Total:";&5,2t
282 rem
284 rem *** CHANGE, RESET, PRINT LABELS ***
286 rem
290 display cl$"Put label roll in printer"
@1,3pr$:wait
300 rem Format for 9 line labels
310 plen 9:tlen 9:space O:print
320 output from "new.inv.list" @5[name]@5[street]
@5[city]@5&[state][zip]cr$cr$cr$cr$ct$
322 rem
324 rem *** CHANGE, RESET, FINISH ***
326 rem
330 display cl$"Put normal stationery in printer"
@1,3pr$:wait
350 plen 66:tlen 60
360 display cl$"Finished"@1,3pr$:wait
370 menu
```

When you have digested the program and understood what it's intended to do, you'll be ready to move on to studying Superbase's reporting functions, which allow you to analyse the data in your files to a much greater extent than the statements we have looked at so far.

CHAPTER 8
REPORTING

Superbase's report generator is invoked by selecting the 'report' option from Menu 1. It then presents you with a series of nine questions about what you want the report to do. When you've supplied the answers, Superbase creates a small program, no different from the ones we've been looking at, in the program area. You then have the option to store the program as it is on the disk, or edit it first. In this chapter I'm going to assume that we are programming a report from scratch, just as if we had selected 'prog' from Menu 2. The report generator is a little difficult to understand unless you have some understanding of the group of statements involved, and I think most users will appreciate it better after learning to program rather than before. So we shall begin with a review of the six special statements that together constitute the reporting function.

The Report Statements

The idea of reporting has two aspects: the production of well presented printout, and the analysis of data. Between them, the six report statements take care of these two requirements. The first two, 'report' and 'endreport', are there mainly to define the beginning and end of a report program. The next, 'title', gives you the ability to print headings on every page. 'Subtotal' is for use with sorted lists, and lets you print out totals for subgroups of records. 'Total' counts and accumulates totals for fields in the record format. And 'detail' actually prints the lines that comprise the body of the report.

Defining Report Style Output: 'report'

'Report' indicates that the other statements in the group are active. It is always the first statement in the group, though it need not be the first in the program. 'Report' in fact acts just like 'file', in that it selects a file from the database catalog and if necessary finds the file definition on the disk and loads it up, opening the file in the process. Like 'file', 'report' requires the name of the file to be used, inside double quotation marks:

```
10 report "invoices"
```

Terminating Report Style Output: 'endreport'

This statement signifies to Superbase that the other statements in the group are inactive. The three functions described next are disabled. 'Endreport' is always the last statement in the group, though it need not be the last in the program. It does not deselect the current file, which remains the one opened by 'report'.

'Endreport' also acts as an output statement. You can place any of the elements usually associated with 'output' after it, and position and format them using the normal commands. This allows 'endreport' to be used to print out the totals that have been accumulated during the print run by the 'total' statement, as well as to print any standard messages such as "End of Report":

```
10 report "invoices"
.
.
.
.
100 endreport "Final Totals are:"t1;t2;t3 plus
110 @1,0"End of Report"
```

Headings on Every Page: 'title'

For a report to look acceptably professional, it must have at least one line of heading on each page. Superbase allows as many lines of heading as you want. They are all described with the 'title' statement, and when a report is being printed Superbase prints the headings you specify each time it starts a new page, the exact point at which a page starts being determined by the 'plen' and 'tlen' statements. (Actually, if 'report' has been executed, the heading specified in a 'title' statement will print out each time there is a page change, regardless of whether the other 'report' commands are in use.)

I cannot recommend too strongly that you design your reports on paper before you begin to code them. You can buy layout forms from office stationers which show the standard width of 132 columns (this width is derived as the result of 11 inches printed at 12 characters per inch, I believe), in zones of 10 columns. All you have to do is position your headings as you want them to appear, whether your printout is to be at 80 or 132 columns width, and work out the starting column position of each item. This then becomes the positioning parameter in the 'title' line or one of the other lines of the report. So, if your heading "SALES REPORT" is to appear at column position 34, the line looks like this:

```
20 title @34"SALES REPORT"
```

This general method of positioning elements of the headings benefits from some specific guidelines:

1. Multiple lines of heading are obtained by inserting blank lines when required. You can use 'plus' to extend the 'title' statement if necessary.

2. Insert blank lines either with a variable set to chr$(13) -- a carriage return -- or the command '@1,0':

```
20 title @34"SALES REPORT"@1@1,0"CUSTOMER NAME"
```

   or

```
15 cr$=chr$(13)
20 title @34"SALES REPORT"cr$@5"CUSTOMER NAME"
```

3. For column headings, align the start of the heading for a text or date item with the left of the item, but for numeric items align to the right of the item:

```
ACCOUNT                BALANCE
jackson01                34.50
```

4. Use dashes to make your column headings easier to read:

```
------------- BALANCES -------------
        MONTH           YEAR TO DATE
       456.89                2000.89
```

5. Use the Superbase underlining commands, '@-' and '@+', to switch underlining on for all or individual items (provided your printer supports the feature):

```
20 title @34@+"SALES REPORT"@1@1,0@+"CUSTOMER NAME"
```

   This also produces reverse video for screen displays, whether you are using report statements or not.

6. People often like the main heading to be central on the page. If you prefer, you can use this simple formula, which involves setting the words of the heading into a variable:

```
15 hd$="CUSTOMER LIST"
20 hp=(80-len(hd$))/2:rem 80 is report width
22 rem hp is horizontal position for any length title
25 title @hp;hd$
```

7. There is no provision for footings as such but you can cheat by making the first line of your title into a footing line and adjusting the page break so that it comes next, followed by the actual title:

```
10 title @34"End of page footing" @1@1,0 plus
20 @34"SALES REPORT"
```

8. Page numbering is not provided directly, but there is a reliable method of achieving this with a slightly complex use of the 'total' statement in conjunction with 'title'. I explain this in Chapter 10.

Counting, Totalling, and Subtotalling: 'total' and 'subtotal'

Although these two statements have very different functions, I am discussing them together because of their inextricably close relationship in practice. Let's consider them briefly in turn.

'Subtotal' is essentially an output statement that is acted on only under certain conditions. It allows Superbase to detect the end of a group of records, such as all those for a particular customer or city. It does this by checking the contents of a field in the record each time it reads a new record. (In fact it only checks the first 16 characters.) You simply specify the name of the field to be checked:

```
10 subtotal [customer]
```

This is known as the subtotal break field. You would usually follow the subtotal break field with a printed message that includes the amount for the group of records just printed, for which [customer] was the same:

```
10 subtotal [customer] "Subtotal for above group:"s1
```

The variable s1 is one of the special subtotal variables which are controlled by the 'total' statement. It will be set to accumulate the value you want to print out for each group, say the customer balance. After s1 is printed when a change in [customer] occurs, it is set to zero, and then starts to accumulate a value for the new group. The changes occur like this:

	Customer	*Balance*	*Value of s1*
	jones	34.56	34.56
	jones	123.45	158.01
	jones	23.50	181.51
*	smith	45.00	0.00

* 'Subtotal' detects the change in [customer], prints out 181.51, sets s1 to zero, and adds the new balance for Smith:

smith	45.00	45.00

'Total' does the counting during a report, and provides the variables to be printed out by 'subtotal' lines. Basically, 'total' allows you to decide what is to happen each time Superbase outputs a record. Your options are limited to assigning values to variables, but there are still many powerful functions at your disposal:

1. The total variables. These are the variables t0, t1, t2, t3, t4, t5, t6, t7, t8 and t9. You specify a field to be accumulated throughout the report, and Superbase adds the contents of the field from each record it outputs:

   ```
   10 total t1=t1+[balance]
   ```

   At the end of the report, the value of the variable is the total amount for the field. Normally, you print it out with the 'endreport' statement (see above).

2. The subtotal variables. These are the variables s0, s1, s2, s3, s4, s5, s6, s7, s8 and s9. You specify a field to be accumulated throughout the report:

```
10 total t1=t1+[balance];s1=s1+[balance]
```

   (Notice the semicolons, they are essential.) Superbase adds [balance] to s1 each time it outputs a record, until a 'subtotal' line detects a change in a break field, causing Superbase to print s1 and reset it to zero. Superbase then adds the [balance] from the new record to s1, and the process continues.

These are the two main purposes of the 'total' line, but the others are nearly as important.

3. Counting. Any variable can be used to provide an overall count of the records output during a report. This is done by adding one to it on each pass:

```
10 total c=c+1
```

   To count subgroups, you must use one or more of the subtotal variables s0 to s9, as these are the only ones that can be reset by a 'subtotal' line.

4. Averages. Once you have obtained a count, you can obtain averages to be printed out in either a 'subtotal' or 'endreport' line:

```
10 total t1=t1+[balance];c=c+1;s0=s0+1;s1=s1+[balance]
20 subtotal [customer] "Records in group:"@40s0 plus s1
21 rem See CAUTION below to explain s1 on above line
25 @1"Subtotal balances:"@40s1 plus
30 @1"Average subtotal balance:"@40s1/s0
.
.
.
80 endreport "Number of records:"@40c plus
85 @1"Total balances:"@40t1 plus
90 @1"Average balance:"@40t1/c
```

5. Retaining the subtotal break field. The 'total' line is mainly used for numeric accumulation. However, you can also use it to assign values to string variables. A common use for this is to keep track of the contents of the subtotal break field:

```
10 total cu$=[customer];s1=s1+[balance]
```

   Superbase assigns the value of the [customer] field to cu$ each time a record is output. It does this only after acting on a 'subtotal' line; this means that if a change in [customer] is detected, the old value of [customer] is still available for the 'subtotal' line itself:

```
20 subtotal [customer] "Subtotal for customer"cu$@40s1
```

The actual printout might look like this:

```
        Customer                                Balance

        Jones                                     34.56
        Jones                                    123.45
        Jones                                     23.50

        Subtotal for customer Jones              181.51

        Smith                                     45.00
        Smith                                    678.89

        Subtotal for customer Smith              723.89

        Williams                                  90.00

        etc.
```

CAUTION. *Even though you can use 'plus' to extend a 'subtotal' line, you must always make sure that any subtotal variables that are to be reset to zero appear on the first line. They may appear after the 'plus' -- the only time this is allowed -- but won't be printed until their next mention:*

```
20 subtotal [customer] plus s1
25 "Subtotal for customer:" s1
```

Printing Record Details: 'detail'

This statement is very similar to 'output'. It works by reading records from a database file, either using an index key list or the whole file as a source. Each time a record is read, the fields you have specified, or other elements such as variables, text, or formulas, are output to screen or printer, depending on which is in force from an earlier statement. So a typical 'detail' line could look like this:

```
30 detail from "sortlist"[customer][balance]
[telephone]
```

As with 'output', you can use any of the formatting or positioning commands, and the 'plus' statement to extend the line. You can specify 'across' or 'down', 'display' or 'print'. The one thing you can't do is output to a disk file, at least not directly (for the Commodore 128 a special statement has been introduced to allow this, and there is a roundabout way of doing it with other Commodore machines.)

One important difference between 'output' and 'detail' is that with the latter you can omit the 'all' or 'from "listname"' that is compulsory with the former. A line like this is acceptable:

```
30 detail [customer][balance][telephone]
```

This line would output only the details from the current record; it would be up to you to select a record from the file in another part of the program, probably using 'select next' or 'select from "listname"'. This allows you to perform any kind of processing in between selecting the record and printing details from it. Often this processing will take the form of checking to see that the record meets certain criteria: a way of combining searching and reporting in one operation. I give an example of a program that does this in Chapter 10.

How the Report Statements Work Together

The key report statements work together in a report program to ensure that presentation and analysis, the two aspects of the operation, are coordinated. After the 'report', 'title', 'total', and 'subtotal' statements have set up the parameters for the program, this is the sequence of events:

1. 'Detail' is the trigger for the other statements. Read a record
2. If the line count exceeds the number permitted by 'tlen', execute 'title' on a new page.
3. If any of the subtotal break fields are different in the current record from their values in the previous record, execute the appropriate 'subtotal' line.
4. Output the elements specified in the 'detail' line.
5. Execute the 'total' line .
6. Read the next record, and continue processing until ....
7. At the end of the file or list, execute the 'endreport' line.

Example Report Program 1

This program adapts the example I used to summarize the previous chapter:

```
10 rem Program name "invoice-1"
12 rem
14 rem *** SET UP INITIAL VALUES ***
16 rem
20 cl$=chr$(147):rem To clear the screen
24 cr$=chr$(13)
26 pr$="Press return when ready":rem Used often
28 h1$="CREDIT CONTROL REPORT":p1=(80-len(h1$))/2
30 h2$="NEW INVOICE SUMMARY":p2=(80-len(h2$))/2
32 file "invoices
42 rem
44 rem *** FIND AND SORT OVERDUE INVOICES ***
46 rem
50 find "inv.list" where [balance] is ">0";[status] is
"=printed"; [date] is ">30NOV85&<01JAN86"
60 sort from "inv.list" on [name][date][sort-1]
to "inv.sort"
```

```
72 rem
74 rem *** SET PARAMETERS, PRINT REPORT ***
76 rem
80 print:across:space 1:cont 1
85 report "invoices
90 title @p1;h1$cr$@1"NUMBER NAME       DATE
TOTAL    BALANCE  TELEPHONE"cr$
91 rem
92 rem *** TOTALS, SUBTOTALS, COUNTS, CUSTOMER ***
93 rem
95 total t0=t0+1;t1=t1+[total];t2=t2+[balance];
s0=s0+1;s1=s1+[total];s2=s2+[balance];cu$=[name]
96 rem
97 rem *** AT BREAK, DO SUBTOTALS, AVERAGES, COUNT ***
98 rem
100 subtotal [name]"Subtotals for customer"
cu$plus s0 s1 s2:rem *** CLEAR SUBTOTALS ***
105 @47&5,2s1 &5,2s2 plus
110 @1"Averages for customer"cu$@47&5,2s1/s0
&5,2s2/s0 plus
115 @1"Number of invoices"@47&2,0s0
116 rem
117 rem *** PRINT RECORD DETAILS ***
118 rem
120 detail from "inv.sort" @1[number][name][date]
&5,2[total]&5,2[balance][telephone]
121 rem
122 rem *** AT END, PRINT ANALYSIS OF DATA ***
123 rem
125 endreport @1,0"Grand totals" @47&5,2t1 &5,2t2 plus
130 @1"Overall averages" @47&5,2t1/t0&5,2t2/t0 plus
135 @1"Overall number of invoices"@47&2,0t0 plus
140 @1"Gross amount paid"@47&5,2t1-t2plus
145 @1"Percentage unpaid"@47&5,2(t2*100/t1);"%"
150 rem
152 rem *** FINISH ***
154 rem
160 plen 66:tlen 60:space 0:cont 0
165 display c1$"Finished"@1,3pr$:wait
170 menu
```

Many of the lines in the program are included purely to document the different sections of the code. When you remove them, you can see that the main part of the report is accomplished in very few lines, fewer still if you make allowances for the use of 'plus'. In the 'endreport' section, I have deliberately included two extra calculations to illustrate the possibilities of formulas as part of the report. You can of course use them in other places, such as the 'subtotal' and 'detail' lines.

PROGRAMMER'S CHALLENGE. *What is the formula to compute a ratio of amount billed to amount paid off for each customer, and how would you print it?*

Example Report Program 2

This is a summary report, which takes advantage of the ability to use 'detail' without specifying any fields or data to be output. The form of the line is simply:

```
10 detail from "sortlist";
```

The semicolon is essential. When this is used, all the other report statements are still active, so totals will be accumulated and subtotal break fields will all be checked even though there is no body to the report. This is the whole point: we can skip the detail, and still obtain useful results. For our example, we'll imagine a sales statistics file with one record for each sale, looking like this:

```
code                <         >
product type        <         >
sales area          <         >
sales amount        <         >
```

A basic format for a simple job. The aim of the report is to produce a summary of sales figures for product types broken down into areas. Here's the report:

```
10 file "stats
20 sort all on [type][area] to "sortlist"
30 report "stats
40 title @34 "SALES SUMMARY"@1@1,0
50 total tl=t1+[amount];s1=s1+[amount];plus
60 s2=s2+[amount];plus
70 ty$=[type];ar$=[area]
90 subtotal [area]@11"Subtotal for area"ar$;@50s1
100 subtotal [type]@21"Subtotal for type"ty$;@65s2
110 detail from "sortlist";
120 endreport @1,0@21"Total amount"@65t1
```

The printout will look like this abstract scheme:

```
Subtotal for area CA   300.00

Subtotal for area NY   300.00

          Subtotal for type A   600.00

Subtotal for area CA   250.00

Subtotal for area NY   250.00

          Subtotal for type B   500.00

          Total amount         1100.00
```

This technique has one big weakness. If the contents of the [area] subtotal break field do not change at the same time as the contents of the [type] subtotal break field, the report will fail to print a final [area] subtotal before the [type] subtotal, and will actually aggregrate the unprinted subtotal with the first subtotal for the new type. This problem can only be overcome by using 'select from' and inserting 'if' statements to check the contents of the fields as they change, which is what you would do in a similar BASIC program.

With this summary report we come to the end of this part of the book, The Automated Database. If you have followed the examples and put some of their principles into practice, you should be able to construct and run a Superbase database using just a small number of tried and tested programs. You will be able to eliminate almost completely the tedium of typing in long command lines, struggling to remember field names, and coping with the inevitable typing mistakes and errors of logic. The next step, when you've reached this stage, is to tie all the small programs together into one big one -- the Programmed Database.

PART III

THE PROGRAMMED DATABASE

CHAPTER 9
PROGRAMMING MENUS AND INPUT SCREENS

This chapter integrates the material of previous chapters, explaining how to set up a menu driven application that brings the major functions of the database together as programs that can be selected and run automatically. In the following chapters I go on to discuss a number of topics of perennial interest to Superbase users, illustrated with examples of more advanced programming. But before I begin the discussion of menus, I must offer some general advice about programming. I do this with serious hesitation, not only because I recognize my own limitations as a programmer, but also because among the many Superbase programmers in the world there are undoubtedly some experts who have their own decided opinions. Well, I am not setting out to offend anyone's preconceptions, just to establish some ground rules for those who need them.

Anyone who wants to program Superbase regularly should treat it as an exercise in learning to use a new programming language. That is what Superbase really is, despite the menus, which are no more than snapshots of some of the main statements shown in a user friendly form. Of course, Superbase is very similar to BASIC, and for this reason I strongly recommend that dedicated Superbase programmers make themselves familiar with that language, preferably a Commodore or Apple version, before getting in too deep with Superbase. Use a good textbook with lots of easy examples. Make sure you know how the following elements of the language function:

Variables:	String, numeric, and array types
Functions:	String and numeric
Operators:	+ - * / ^ = < > and or
Branching:	goto gosub .... return
Loops:	for ... to ... step next
Conditionals:	if ... then ... else

Program structure depends on the purpose of the program, but there are a few fundamentals that are common to most programs of any length. Programs are usually divided into three parts: initialization, main code, and subroutines. There may also be a part for storing data to be used in the program.

The first part of a program is initialization. This refers to such things as setting up any variables and tables (arrays) of data to be used later in the program, opening the files that the program will require, as well as perhaps displaying an initial screen and requesting any critical run time parameters.

Logically, the next part of a program is the main code. This steps through the sequence of operations that accomplish the desired goal, whether this is selection from a menu or the production of invoices. At the end of the main code, the program should exit after ensuring that all is in order in its world of data and variables -- no open files or records that have not been updated.

Any parts of a program that are repeated can be placed in subroutines to which the main code branches when necessary. Subroutines should be structured like small programs within the main program, with their own variables, initialization, calls to other more general subroutines, and carefully controlled exit points. Some programmers believe that the lines containing subroutine code should be placed near the beginning of a program, before the main code section, as this yields a small increase in speed, but as far as Superbase is concerned the difference will be minimal.

Data storage. In a Superbase program, like a BASIC program, 'data' statements are conventionally placed at the end of the program.

Program style is again a matter for individual preference, but these general precepts will, I hope, meet with general approval.

Make the program legible. When possible, space out the different parts of the code with 'rem' statements. Commodore BASIC is very unhelpful when it comes to presenting code, and there is no way to indent 'for ... next' loops, for example, or to keep the syntax properly spaced out on a line.

Comment often. Explain to the reader what each part of the program is doing. You may not be the only person ever to look at the code. Of course, if your version of Superbase allows only 4K of program, you will be tempted to break this rule, especially as shorter programs also load more quickly and take up less room on the disk. Even so, comment where possible, even if the version you use is not the one you print out for your records.

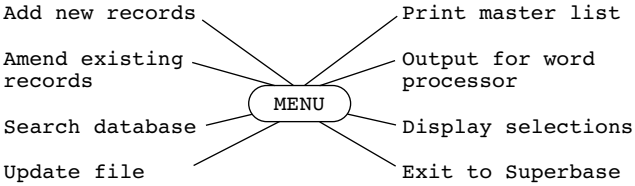
KISS. Keep it simple, stupid. Avoid excessively fancy contortions in the code. They may gratify your sense of intellectual superiority, but they will perplex and infuriate both you and others six months from now when the bug has to be fixed by noon. Try to be satisfied with the elegance of the obvious. Of course, there will be times when results cannot be obtained without a descent into obscurity, or when memory limitations impose terseness.

Flow downhill. Try to keep the flow of program control moving in one direction, from beginning to end, except for those programs that mainly consist of one big loop.

If you consider that the preceding paragraphs are on the whole meaningful to you, you're probably ready to come to terms with the next subject, constructing a menu program.

A Menu Driven System

We can define a menu driven Superbase application as a system in which the main functions are obtained, or accessed, from a single point of overall control. Superbase itself is menu driven, and you can call any of its primary functions from the main menus. Menus offer options to the user, who can select one by entering a number or pressing a letter key. Structurally, a Superbase application menu can be represented as a node:



Each function is a separate program, which is executed when the user selects it from the menu. When the executed program is finished, it reloads and executes the original menu program. On the screen, the menu is more likely to be a list:

```
        MY MAIN MENU

1 Add new records

2 Amend existing records

3 Search database

4 Update file

5 Print master list

6 Output for word processor

7 Display selections

8 Exit to Superbase

Enter option:
```

So, how do we obtain a menu like the above on the screen? There are three main ways of doing it: using help screens, using database file formats, and direct coding.

Method 1: Help Screen

The first method is the simplest. You use the Superbase 'memo' function to design a screen, typing in the descriptions and numbers of the options, a heading for the menu, and any boxes, underlining, etc. that you want (and the keyboard allows). You also type in a prompt, such as the "Enter option:" message above. You then store this memo screen on the disk, and by giving it a name beginning with "h" or "h8" (depending on your version), you make it available to the Superbase 'help' command. This means that a file called "h8menu1" can be displayed on the screen in its entirety with the simple program line:

```
10 help "menu1
```

Notice that I have deliberately omitted the "h8" at the beginning of the name -- 'help' always supplies it before looking for the file on the disk.

Method 2: Database File Format

The second method, using database file formats, is the least obvious. You use the 'format' option to design up to four menu screens, using the drawing facilities as you please and typing in the option numbers and descriptions as above. The four formats are stored as a normal data file, so you need to put one field on each of the screens you design. Use the key field on the first screen and text field replicas of it on subsequent screens.

This file is then selected with a normal 'file' statement, which is accompanied by a 'screen' statement to decide which of the formats to display, and a 'select c' to display it:

```
10 file "menus":screen 0:select c
```

With ingenuity, you can ring the changes on 'screen n' to produce very fast displays of even the maximum of four menus, each with a number of options on it. Each time you display a screen, though, you must handle the single field on it. I usually obscure it by displaying the "Enter option" message over it, having carefully positioned the field in the right place to begin with.

Method 3: Direct Coding

The third method, direct coding, is closest to BASIC. You just use 'display' with the option numbers and descriptions, positioning them at the column and row you want:

```
10 display @30,3"1 Add new records"
20 display @30,5"2 Amend existing record"
30 display @30,7"3 Search database"
```

This is fast, but not as fast as the previous method, and it can consume a lot of memory space, which could often be put to better

use. However, if for some reason you want the bells and whistles of a Superscript style duckshoot menu, you can have them -- provided you've got unlimited time for programming!

Example Menu Program

Overall, the first method is preferable. It involves less programming and the time taken to display a help screen is no longer than that required to load a long file definition. Moreover, the overhead for having one of the current files dedicated to a menu function is unacceptable when a multi-file application is needed. Here is an example of a menu display and selection using a help screen:

```
10 rem Superbase menu program
18 rem
20 rem *** LOAD PROGRAM NAMES INTO ARRAY ***
28 rem
30 for i=1 to 7:read prg$(i):next
38 rem
40 rem *** MENU STARTS HERE ***
48 rem
50 help "menu1"
60 display @0@34,20@+" ";
70 wait op: if(op<1)or(op>8)then 70
80 if op=8 then menu
90 load prg$(op):rem *** END OF PROGRAM ***
98 rem
100 rem *** PROGRAM NAMES FOR LINE 30 ***
102 rem
110 data newrec,amdrec,srcdb,updfile,prtmast,wp,disp
```

This is in effect a seven line program. Line 110 holds the names of the seven programs that correspond to menu items one through seven. Line 30 reads the program names into an array, prg$(n). Line 50 displays the help screen that contains the menu, and line 60 displays a reversed space after the "Enter option" prompt, purely for cosmetic reasons, as the input does not in fact take place at this point. Line 70 does the job of getting a key from the keyboard. Since I have requested a numeric variable, op, no alphabetic key will be accepted. The line also checks to see whether the value of op is within the range of options, one to eight. If it is not, the user is forced to input a new number. Line 80 tests to see whether option 8, exit to Superbase, has been selected; if it has, the program exits. Line 90 ends the program by loading and executing the program from the element of the prg$(n) array that corresponds to the selected option.

Using "start.p" to Display a Menu

It would be convenient to have this menu displayed as soon as Superbase is loaded. This is made possible by the "start" program. Everyone who owns Superbase has used this program, even if only a minority have ever investigated it as a program. It sets up system

parameters like margins and line spacing, draws a box on the screen with Superbase's name inside, and asks for input of a database name and a file name. That's all. Although these functions are useful, they are not essential. The most important thing about the "start" program is its name. Superbase always looks for a program with this name on the disk you insert after the start up menu comes up (the one with "Remove Program Disk", etc). (If you have not placed a copy of "start" on your work disk, Superbase says "File Not Found"; this does not prevent you from selecting a database and file and carrying on normally.)

The implication of the fact that the name is what's important about "start" is that you can modify it to include any statements that you want. You can remove the display of the box and "Superbase", and you can specify a database and a file by name rather than requesting user input. You could even write an entirely new program and call it "start". What sort of program? A menu, of course. Here is an example of a start program that sets the system parameters, selects database and file, and then calls up a menu as in the previous example.

```
10 rem Superbase start program
18 rem
20 rem *** SET SYSTEM PARAMETERS ***
28 rem
30 lmarg 1:rmarg 80:rem margins
40 plen 66:tlen 60:rem page & text length
50 pdev 4:pdef 0:rem printer device 4 cbm code
60 lfeed 0:cont 1:rem no line feeds, continuous print
70 space 0:across:screen 0
74 rem
76 rem *** SELECT DATABASE AND FILE ***
78 rem
80 database "sales":file "customers"
98 rem
100 rem *** LOAD PROGRAM NAMES INTO ARRAY ***
118 rem
120 for i=1 to 7:read prg$(i):next
128 rem
130 rem *** MENU STARTS HERE ***
138 rem
140 help "menu1"
150 display @0@34,20@+" ";
160 wait op: if(op<1)or(op>8)then 160
170 if op=8 then menu
180 load prg$(op):rem *** END OF PROGRAM ***
184 rem
186 rem *** PROGRAM NAMES FOR LINE 120 ***
188 rem
190 data newrec,amdrec,srcdb,updfile,prtmast,wp,disp
```

Extended Menu Programs

Earlier I stated that each of the options on the menu is normally a separate program. It may well be possible, if the code in these

programs is short enough, to incorporate it in the main menu program itself. While this blurs the structure of the menu and its dependent satellites, it allows faster execution, and if one or more of your options is used often, you should certainly consider this move. Here's the first menu program amended to allow database searching and review of the records in the current "h8list":

```
10 rem Superbase menu program
20 rem *** LOAD PROGRAM NAMES INTO ARRAY ***
30 for i=1 to 7:read prg$(i):next
40 rem *** MENU STARTS HERE ***
50 help "menu1"
60 display @0@34,20@+" ";
70 wait op: if(op<1)or(op>8)then 70
72 if op=3 then 200
74 if op=7 then 300
80 if op=8 then menu
90 load prg$(op):rem *** END OF PROGRAM ***
100 rem *** PROGRAM NAMES FOR LINE 30 ***
110 data newrec,amdrec,dummy,updfile,prtmast,wp,dummy
198 rem *** CODE FOR MENU OPTIONS ***
200 find
210 goto 50
300 output from "" [code][name][balance][telephone]
310 goto 50
```

Notice that in line 110 two of the program names have been changed to "dummy", and that following each option the program returns to line 50 to display the menu.

Designing Your Own Input Screens

Once you have become aware of the possibility of programming in your own menus, you will probably start to wonder whether you can do more to program Superbase's screen displays. You certainly can.

Users often want to create their own alternatives to the Superbase menu options 'enter' and 'select replace' or 'select add'. Another common need is for introductory screens, where the operator is asked for some input, such as a password or a date, before the main program proceeds. Superbase has two input statements, 'ask' and 'wait', and one output statement, 'display', which you have already seen at work.

The 'ask' Statement

The 'ask' statement is for inputting data of any length up to 254 text characters. It is self-validating in that if you specify a numeric input variable only numeric input will be accepted. Here are the basic forms of 'ask':

```
10 ask x$
20 ask x
```

These two variants cause the command line to display the prompt "Enter ?"; you must type in a response followed by return. Return on its own is not permitted, neither is a space, but zero may be typed in for the numeric variable. If you try and input a non-numeric variable to x, you will get an error message; when you press return, the program executes the line again.

```
10 ask @10,5;x$
```

This displays a cursor at column 10, row 5; validation is the same. The next example, valid for both x and x$, displays the prompt message on screen, and then accepts the input:

```
10 ask @10,5"Please enter your name: ";x$
```

If you omit the positioning command, the message appears on the command line, preceded by the word "Enter". You can also use 'ask' to input directly into a record format, field by field. The following program shows how:

```
90 clear:select c
100 ask "code"[code]:select c
110 ask "name"[name]:select c
120 ask "street"[street]:select c
130 ask "city"[city]:select c
140 ask "state"[state]:select c
150 ask "zip"[zip]:select c
160 ask "telephone"[telephone]:select c
170 ask "amount"[amount]:select c
180 ask "date"[date]:select c
190 store
```

The first line clears the record format and produces a blank record to be filled in, then shows it on the screen with 'select current'. Each of the following lines inputs one field and displays the record format updated with the new item. (If you input just a string of numbers such as a zip code, Superbase formats them as a number, i.e. with two decimal places and a leading space for the sign. To overcome this anomaly, type in a space before the number.) At the end, 'store' puts the new record into the file. Each item is typed in on the command line, not into the field itself. The program can request fields to be entered in any order. Numeric and date fields are validated automatically, but you can also intervene in the input process to check the contents or range of any field. Suppose [code] may not begin with the digit "2"; the program can be modified like this:

```
90 clear:select c
100 ask "code"[code]:if left$([code],1)="2" then 100
105 select c
110 ask "name"[name]:select c
etc.
```

Input will be forced back to the same line until a legal code is entered.

Be aware of two types of problem. First, you cannot skip fields. Numeric and date fields can accept 0, but text fields must have at least one character. More seriously, you cannot move from field to field to edit the contents, as you can with 'enter' or 'select r', without some rather fancy programming, which is generally not worth the memory space. You have to recall the record at a later time to edit it.

If you prefer, you can include positioning commands to cause the input to be done in the actual field position on the screen. You must also specify the maximum length of the input string, which is done with the '&' command followed by the length of the field:

```
90 clear:select c
100 ask &10@12,2[code]
110 ask &24@12,4[name]
120 ask &24@12,6[street] etc.
```

We dispense with both the prompt string and the 'select c', but there is still no way of editing the fields once you have left them unless you add more code.

A little known feature of the 'ask' statement is that it accepts whatever is at and to the right of the cursor position when return is pressed. This means that you can display some characters on the screen, and then position to the first one and do an 'ask' into a string variable. When the operator presses return the variable will hold all the characters to the right, as well as the one under the cursor. Try this short demonstration program:

```
100 display @5,10"This is the string to be input"
105 display @5,11"(Press return)
110 ask @5,10x$
120 display @5,13"x$="+x$:wait
```

Clever programmers use tricks like this for selecting files or options from a display, but it is not an essential part of the 'ask' repertoire.

The 'wait' Statement

The difference between 'wait' and 'ask' is that the former only accepts single key input, whereas the latter can accept strings up to 254 characters long. 'Wait' is thus used for the ubiquitous "Press any key to continue" routine that inserts a pause in the execution of a program, usually so the operator can read a screen message. In Superbase, whenever you see the message "Waiting" on the command line you know that a program is executing the 'wait' statement.

```
10 wait a
20 wait a$
30 wait
```

The first line accepts only numeric input; we used it earlier in the menu program. The second line accepts any key, and preserves the value in the variable a$. The third line accepts any key, but does not preserve its value, and is the most suitable when a pause pure and simple is required. Line 20 allows you to trap the input key, test its value, and act accordingly. Another use is to provide a quick way of ascertaining the ASCII value of the key:

```
20 wait a$:display asc(a$):wait
```

'Wait' does not display the input key. This makes it suitable for accepting passwords. Here is a program that allows the user a number of tries at a password before denying access:

```
20 at=3:rem number of attempts
30 ln=5:rem length of password
40 for j=1to at
45 x$=""
50 display chr$(147)@5,5"Enter password:";@+" ";
100 for i=1to ln
110 wait a$
120 x$=x$+a$
130 next i
140 if x$<>"super"then display "Failed!";:goto 150
145 display "OK!":goto 180
150 display @5,10"Attempt";&1,0j;"of";&1,0at;"failed"
155 wait 160 next j
170 display @5,12"Attempts failed -- access denied!"
175 quit
180 load "menu"
```

The j 'for ... next' loop controls the number of attempts, while the i 'for ... next' loop accepts five successive key inputs, building up the full password in x$. This is then compared in line 140 with the correct password, and the program acts appropriately depending on the results. If you use the 'protect' statement (see Manual) on a password program, you can stop anyone looking at it. This includes yourself -- so ensure you have the unprotected original locked away safely.

Another common use of 'wait' is to obtain the response to what I call a confirmation prompt, often taking the form "Are you sure? (y/n)". Here the job is to verify the input as quickly as possible:

```
100 display @1,22"Are you sure? (y/n)"@0;:wait a$
110 if a$<>"y"and a$<>"Y"and a$<>"n"and a$<>"N" then 100
```

The lengthy line 110 can be replaced with the preferable 'instr' function if this is available in BASIC:

```
110 if instr("YyNn",a$,1)=0 then 100
```

The 'display' Statement

'Display' is very widely used in screen handling programs. It allows precise positioning of characters and truncation of text strings, as well as the reverse video effect. Its effect is controlled by the 'across' / 'down' pair of statements, so if you start to see "End of page" during the display check this first. Here are some of the possible forms 'display' can take:

```
10 display "John Smith";
20 display @+"John Smith"
30 display @1,14"John Smith"
40 display &4"John Smith"
50 display @1,14@+"John Smith"
60 display @0@1,14"John Smith"
```

Line 10 does a straightforward display, but the semicolon at the end stops the cursor from moving to the next line, so line 20 shows a reverse display on the same line. Line 30 uses column and row positioning, and line 40 shows how to truncate the display. Line 50 illustrates how attempting to do a display at a position that is prior to the current cursor position causes an "End of page", while line 60 shows the correct way to reposition on the screen without forcing the new page. The '@0' positioning command in line 60 should always be used in conjunction with the other coordinates to reset the display counter (an internal event) if you want to display above or to the left of the cursor position without disrupting what's already on the screen.

Drawing Boxes

In a fully programmed display, users often want to use boxes to emphasize parts of the screen. 'Display' can be used to draw such boxes, but the code is like BASIC and there are no special windowing statements:

```
10 rem drawing boxes
15 rem w width - h height - r row - c column
14 rem
16 rem *** PARAMETERS FOR 6 BOXES ***
18 rem
20 w=20:h=10:r=5:c=10:gosub 60:b$="1":gosub 90
25 w=4:h=3:r=2:c=34:gosub 60:b$="2":gosub 90
30 w=10:h=2:r=18:c=3:gosub 60:b$="3":gosub 90
35 w=6:h=6:r=8:c=70:gosub 60:b$="4":gosub 90
40 w=15:h=8:r=12:c=40:gosub 60:b$="5":gosub 90
45 w=30:h=1:r=4:c=45:gosub 60:b$="6":gosub 90
50 wait:menu: rem **** END OF PROGRAM ****
52 rem
54 rem *** BOX DRAWING SUBROUTINE ***
56 rem
60 h$="":for i=1to w-2:h$=h$+"-":next
65 display @0@c,r;"┌"+h$+"┐":rem TOP CORNERS
70 for i=1to h:display @c,r+i;"│"@c+w-1;"│":next
75 display @c,r+h+1;"└"+h$+"┘":rem BTM CORNERS
```

```
80 return
82 rem
84 rem *** TEXT IN BOX SUBROUTINE ***
86 rem
90 display @0@c+1,r+1;b$:return
```

It may not be the fastest thing you ever saw, but it does the job. Lines 10 to 45 simply pass the size and position parameters for the six boxes. Lines 60 to 75 build up a horizontal bar in h$, display it with corners added, display the sides in a 'for ... next' loop, then display the bottom bar and corners. You can substitute your own characters for the Commodore graphics I have used if you prefer. Line 90 shows how to position a text item inside the box after it's been drawn.

Error Messages

In this category I include all text messages that your program puts on the screen to help the user understand how to act when a mistake has been made, but not of course Superbase's own error messages, which are displayed on the command line. This includes both complex errors and simple ones like "Password incorrect". It is important to keep your programs predictable, so decide on a particular area of the screen that you want to reserve for messages, and stick to it in all programs. Error messages are best handled by a subroutine, like this:

```
1 rem *** SET UP SPACES TO CLEAR LINE ***
5 rem
10 sp$="                              ":sp$=sp$+sp$+sp$
14 rem
16 rem *** EXAMPLE ERROR MESSAGES ***
18 rem
20 e$=" Please type six characters ":gosub 60
30 e$=" Date must be in form ddmmmyy ":gosub 60
40 e$=" Range is 1 to 9 ":gosub 60
50 wait:menu
54 rem
56 rem *** ERROR DISPLAY SUBROUTINE ***
58 rem
60 display @0@1,22@+e$@56"Press return to continue";
70 wait:display @0@1,22sp$;@0:return
```

The routine puts the message on the bottom line of the screen, showing it in reverse video for emphasis. Line 10 has the job of building the sp$ string that is used in the subroutine to clear the line after the message has been displayed -- an essential feature that is sometimes overlooked.

CHAPTER 10
ADVANCED PROGRAMMING

This chapter is dedicated to the unknown Superbase user, the intrepid traveller who has struggled through despair to attain that tenebrous object of desire, the perfect program. Some of the solutions on offer in the following pages will certainly be known to some users, but not to all. Likewise, some of them will be inferior to those developed by others, for programs can always be improved; but if our answers enlighten just one darkness, they have served their purpose.

Searching with User Input Criteria

The requirement is for a program that allows the user to input data which is then used in a 'find' operation. The problem is in constructing the where clause so that the input data is joined with the operator, such as or '&', so that Superbase can understand it. Basically, the operator is placed inside quotation marks, and the variable is concatenated with the '+' sign:

```
10 find "" where [name] is "="+nm$
```

If nm$ were equal to "jones", this would be the same as saying:

```
10 find "" where [name] is "=jones"
```

You can use any of the operators in this way. However, there are a number of complications, especially where numeric and date fields are concerned. First, let's see what you have to do for a numeric field:

```
10 ask "amount";x
20 find "" where [balance] is ">"+str$(x)
```

You input the amount into a numeric variable, x; this ensures it is a valid number, not a string. Then, for the 'find' line, you must treat x as a string, because this is all that 'find' understands when it's looking at the line. Use the 'str$' function, as shown above. This is a little confusing, as you can still type a number directly into the quotation marks in a 'find' line:

```
20 find "" where [balance] is ">"+str$(x)+"&<1000"
```

But that's the way you have to do it.

Dates are more complicated still. You have to type the date into a string variable, and this means that you must validate it before you use it in a 'find' line. Use the 'date' statement, which returns zero for an invalid date and the month number for a valid one:

```
10 ask "date";dt$
20 date dt$,m:if m=0 then 10
```

This ensures that only a good date is entered, so the search can go ahead:

```
30 find "" where [date] is "="+dt$
```

A frequent problem is the need to have the operator enter a date as the basis for a search, say "1FEB86", when in fact it is all dates after and including the entered date that are wanted. As Superbase does not have the '>=' operator for dates, it is up to the program to reduce the input date -- a string -- by one, to "31JAN86" in the example, so that the search will catch all dates after that. This involves two extra steps: assigning the string to a date field, which is capable of being manipulated arithmetically; and reconverting the date field to a string for the search. (In this case we are using a date field from the record format purely to assist in the program, not in order to store information in it.) We must also ensure that the input string is seven characters long (ddmmmyy or mmmddyy):

```
10 ask dt$
15 if len(dt$)<>7then 10
20 date dt$,m
25 if m=0then 10
30 [date]=dt$:[date]=[date]-1:convert [date],dt$
40 find""where [date]is">"+dt$
```

A similar technique can be used to find a range of dates between and including two input dates. However, a more sophisticated approach is needed if we want to make the operator's life easier by asking for only the three letters of the month when a search for all the dates in a particular month is called for. It's a little clumsy to use an array of month abbreviation strings and the number of days in each month when Superbase's own facilities can do the job just as efficiently. Here's the program:

```
100 ask &3"month abbreviation";m$
102 rem
103 rem *** VALIDATE DATE, GET MONTH # ***
104 rem
105 if len(m$)<>3 then 100
110 m$="01"+m$+"86":date m$,m:if m=0then 100
112 rem
114 rem *** GET TWO DATE NUMBERS ***
116 rem
120 [date]=m$:d1=[date]-1:d2=[date]+31
122 rem
123 rem *** LAST DAY OF PREVIOUS MONTH ***
124 rem
125 convert d1,d1$
126 rem
127 rem *** FIRST DAY OF NEXT MONTH ***
128 rem
130 d2=d2-1:convert d2,d2S:date d2$,m1
135 if m1<>m then 130
140 d2=d2+1:convert d2,d2$
```

```
142 rem
144 rem *** SEARCH ***
146 rem
160 find""where [date]is">"+d1$+"&<"+d2$
```

First we get a three character identifier ("feb"), then we add a day and a year and validate it ("01feb86"). This is m$, and the month number is m. In line 100 we set up two numeric dates, d1 and d2, respectively one less and 31 greater than the original m$. By adding 31 we ensure that d2 is a date in the next month; exactly which one does not matter. Then we use 'convert' to make d1 into the last date of the previous month, d1$ ("31jan86"). Now, we have to make d2 equal to the first date in the next month (i.e. "01mar86"). In line 130 we decrease d2 by one, 'convert' it, and get its month number, m1, with 'date', which in line 135 we compare with the original, m. If necessary these steps are repeated. When m1 and m are the same, we're on the last day of the original month ("28feb86"), so we just increase d2 by one and it's equal to the first day of the next month. After using 'convert' once more we have the two dates in string form, d1$ and d2$, "31jan86" and "01mar86", ready for use with 'find'. It's more complicated to describe than it is to program, but it works quite satisfactorily.

Programmed Selection of Records

Record selection is one of the more frequently used Superbase options, and in programs there are two main types of requirement: reproducing the functions of the 'select' submenu, and selection by key from an index key list created with 'find' or 'sort'. We'll look at the 'select' submenu functions first. Although you can use 'select' by itself as a program statement, this is rarely satisfactory; Superbase does not allow you to stay at the submenu level without special interpretation of the keyboard input. Therefore you will need code that can use the submenu functions flexibly and effectively.

The single most important function is selection by index key, which allows you to call up a record within one or two seconds. This is very useful, and can be made even more convenient when the input, selection, and display are programmed in.

The first requirement is simply to accept a key string and find a record. For this 'select k' is sufficient:

```
10 select k:select c:wait
```

You are asked to enter a key; Superbase then finds the record, but it is only displayed when 'select c' is executed, and unless 'wait' delays it, the display will be momentary before Superbase returns to the menu.

This is fine, but there are times when you want either to validate the key input or to position the input elsewhere than on the command line, which is where 'select k' is restricted to. For this we just add 'ask':

```
10 ask @5,5"Please enter index key: ";k$
20 if k$="illegal" then 10
30 select k$:select c:wait
```

Line 20 is meant to illustrate the possibilities for validation.

Another requirement for selecting is to call up the last in a series of records with a sequence of keys such as:

```
smith01
smith02
smith03
smith04
smith05
smith06
```

The aim is often to find out the value of the last numeric suffix so that a key can be made for a new record that does not duplicate an existing one. The method involving the fewest reads of the file is this:

```
10 ask @5,5"Please enter name: ";k$
20 k$=k$+"999"
30 select k$:select p:select c:wait
```

The user enters the name, but no sequence number, and the program supplies "999". 'Select k' using this string gets the next record after the last one in the target sequence, which a 'select previous' obtains as the current record ready for display. This code can be combined with programmed record entry to generate keys automatically, avoiding tedious manual searching for the last record in a sequence.

What happens if the key you input does not have a corresponding record in the file, or one which corresponds only in part? Superbase has two special statements for testing the validity of a 'select k' operation: 'nmat' and 'pmat'. The first, 'nmat', is "true" if Superbase cannot find any key that matches. If the index keys were all for "smith" as above, and I entered "williams", 'nmat' would be true. Like any conditional statement, 'nmat' only allows the statements to the right of itself on the same program line to be executed when it is true:

```
10 k$="williams"
20 select k$
30 nmat display "No key for";k$:wait:menu
```

The second test, 'pmat', is true only if Superbase finds a partial match; 'pmat' acts the same way as 'nmat'. A partial match needs defining carefully. If the keys are all for Smiths, as above, the following entries would all produce a condition of true for 'pmat':

```
smith0
smith
smit
smi
sm
```

However, all the following would make 'pmat' false:

```
smith1
smitha
thompson
t
```

A program that performs a full test on 'select k' could look like this:

```
10 ask "key";k$ 20 select k$
30 pmat ms$="Partial";:goto 100
40 nmat display "No key":wait:goto 10
50 ms$="Found";
100 display ms$:"key -- Press return"
110 wait:select c:wait:menu
```

Slightly different code is needed depending on whether you are searching to see whether a record is or is not present in the file. In the above example, line 50 acts on the assumption that the key is found, after ruling out the other two possibilities. To achieve the other objective, the program would be recast like this:

```
10 ask "key";k$
20 select k$
30 nmat ms$="No key":goto 100
40 pmat ms$="Partial key":goto 100
50 display "Key exists":wait:goto 10
100 display ms$;"exists -- Press return"
110 wait:select c:wait:menu
```

It may be that selecting by key is only one of the options that are required. You may also want to browse in the file, change from screen to screen, or take other action such as appending the key of the record you're looking at to a list for output later on. This example shows a way of programming the browsing operation; I shall cover record key picking later.

First we present a blank record format:

```
10 clear:n=0:mx=3:select c
```

The variable mx is the maximum number of screens for this file; n is the current screen. Now 'ask' is used to show a list of the options on the command line, resembling the 'select' submenu itself:

```
20 ask&1"k/f/n/p/l/q/+/-";op$
```

The "q" option is so we know when to quit. 'Wait' cannot be used as it always clears the command line. Next we validate the input with a very useful little routine that applies in many similar situations:

```
30 e=0:rem Flag to register OK when 1
40 m$="kfnplq+-":rem All legal keys
50 for i=1 to len(m$)
60 if mid$(m$,(i-1)*1+1,1)=op$ then p=i:e=1
70 next i
80 if e=0 then 20:rem Key was not valid!
```

(Normally such a routine would be a separate subroutine for use by other pieces of code, with m$ and op$ as standard parameters.) Now we can act on the variable p, which registered the position in the test string of the selected option:

```
85 on p goto 90,100,110,120,130,140,150,160
90 select k:goto 200
100 select f:goto 200
110 select n:goto 200
120 select p:goto 200
130 select l:goto 200
140 menu
150 n=n+1:if n>=mx then n=mx:goto 200
160 n=n-1:if n<0 then n=0
200 screen n:select c:goto 20
```

Lines 90 through 130 execute a 'select' option, and jump to 200 which displays the current screen and returns to the top for the next selection. Lines 150 and 160 control the selection of screen within a multi-screen format, using the variable n for the screen number (0 through 3). The variable mx holds the maximum number of screens for the current file.

Selecting from a List: 'select from'

One of the disadvantages of the 'output' statement is that while it is able to read records using an index key list there is no way to intervene in the process. The statement reads and outputs all in one go, if you like. And yet there are inevitably times when you want to do things like check whether a certain field in a record is empty, or perform calculations, or look up a value in a memory array, before the output takes place. For this, Superbase has the statement 'select from "list"'. It is quite straightforward:

```
10 select from "h8list":eol 100
20 if [code]="smith01" then 10
30 if [balance]<1 then 10
40 detail [code][name][balance][telephone]
50 goto 10
100 display "Finished":wait:menu
```

Line 10 selects the first record from the list, making it the current record. The 'eol' statement acts in the same way as the 'eof' conditional, testing at each selection to see whether the end of the list has been reached; if it has, control passes to line 100. Lines 20 and 30 represent the opportunities for processing the records provided with 'select from'. Both of them cut short the process and go back to get the next record. Line 40 uses 'detail' to output the required fields. Note especially that 'detail' is here used without either 'all' or 'from "list"'. This causes it to refer to the current record only, namely the one read by 'select from' four lines earlier. Line 50 returns control to line 10 to read the next record.

Multi-file Applications: 'link'

Although Superbase is far from being a relational database, it does have the ability to use any item of data from a record in one file as the index key for a record in another file. This function is sometimes described as semi-relational.

The benefit of being able to link from one file to another lies mainly in the economies of disk space that can be achieved, but also in the fact that programs can easily switch from one file to another to update it. An example of both benefits together would be a system in which the creation of an invoice record updates both the customer record balance and the inventory file, and the invoice printing program reads the customer's name and address from one file and the invoice details from another. In short, the database can be made smaller and faster.

There is a group of four statements which control the different aspects of linking. We'll start by taking a look at what they do:

setlink This specifies the link between one file, sometimes called the main file, and another. You are assumed to be in the main file, "file1", and you set a link to another, "file2":

```
10 file "file1":setlink "file2"
```

elink Undoes the link. When a link is in use, Superbase switches between the two files, and if you interrupt the program you may end up with the 'setlink' file as the current file. Superbase shows this by displaying '+' instead of '=' to the right of the File Selected message on the main menus. 'Elink' cancels the link, but if '+' is showing you will find that the 'setlink' file becomes the current file. It's a good idea to execute an 'elink' at the start of a program that uses linking, especially when you're developing it, to cancel any links inadvertently left in place:

```
5 elink
10 file "file1":setlink "file2"
```

link This switches from the main file to the 'setlink' file and attempts to find a record in it. The key for the search, which is just the same as a 'select k', is specified before 'link' is executed. The key can be specified as either a field or a string variable; if it is omitted, Superbase uses the index key of the current record as the default:

```
10 link [cust.ref]
10 link k$
10 link
```

rlink When you have finished processing the 'setlink' file, 'rlink' returns to the current record in the first file.

There are two main reasons for linking between files. Either you want to look something up from another file, or you want to add a record to it. Let's consider first a simple example of looking up data.

```
10 file "invoices":setlink "customers"
20 ask "invoice number";k$:select k$
30 link [cust.ref]
40 display [code][name][telephone][balance]:wait
50 rlink
60 goto 20
```

The program obtains an invoice number in k$, uses this as an index key, and reads an invoice record. It then uses one of the fields in the invoice record, [cust.ref], as the index key for the linked file of customer records. On finding the customer record whose key field, [code], is the same as [cust.ref], the program displays some of the record details. Then it returns from the link in line 50, and loops back to line 20.

CAUTION. *Whether you use a field or a variable as the linking element, it must not be empty, or you will see the message "Invalid FMS Parameter".*

You should really use 'nmat' and 'pmat' to ensure that none of the operations are invalid. These commands are equally useful with both 'select k' and 'link', which function in the same way:

```
10 file "invoices":setlink "customers"
20 ask "invoice number";k$:select k$
22 pmat 20
24 nmat 20
30 link [cust.ref]
32 pmat 50
32 nmat 50
40 display [code][name][telephone][balance]:wait
50 rlink
60 goto 20
```

Now let's reverse the example, and suppose that the application requires us to use a reference in the customer record to call up

all the invoices for a particular customer. This is a classic master and transaction file relationship. The program assumes one critical fact: that all the index keys for the invoices contain the reference from the customer record as the first part of the key:

```
Reference from customer record:          smith01

Invoice number (index key):              smith01.001
                                         smith01.002
                                         smith01.003
                                         smith01.004
                                         smith01.005
```

Now we can use an 'if' statement to test whether all the invoices for a particular customer reference have been read and displayed:

```
5 elink
10 file "customers":setlink "invoices"
20 ask "customer code";k$:select k$
22 pmat 20
24 nmat 20
30 ir$=[inv.ref]:f=len(ir$)
35 link ir$
40 nmat display "No invoices":goto 60
45 if left$([number],f)<>ir$ then 60
50 display [number][date][total][balance]:
select n:goto 45
60 rlink:wait:goto 20
```

Line 30 sets [inv.ref] into a variable, ir$. Variables are often required in link processing, as the fields from one file cannot be referred to while the other is current. Variables, however, are independent, and are suitable for holding data from the one file which is needed in the other. Here, the link to the invoice file is made in line 35. Once into the invoice file, the program loops from line 50 to line 45, alternately testing the validity of the current record against the original invoice reference in ir$, and displaying details prior to reading the next record in sequence in the file. The variable f measures the length of ir$, and ensures that the correct portion of each new invoice number is tested against ir$. Once a record whose key does not match ir$ is detected in line 45, the link is terminated and the program loops back to line 20 to ask for another customer code.

As it stands, this program is pretty basic. But the addition of some more display statements and use of reverse video could soon convert it into a very useful inquiry routine.

The other major requirement for linked file processing is to be able to create records in a main file and a linked file. Here the logic of the program is determined by the fact that when Superbase returns from a link it comes back to the current record, and a blank record that has had data entered into it does not count as a current record until it has been stored. Consequently the program must create the main file record after the link file record. The plan for the program is this:

```
1. Select main file
2. Assemble link key
3. Link to linked file using link key
4. Create record: 'clear'; enter data; 'store'
5. Return to main file
6. Create record: 'clear'; enter data; 'store'
```

In step 3, we are concerned with the results of the 'link' in case a record with the new key already exists. 'Nmat' and 'pmat' should be used to test for this condition.

A common variation on this scheme is the program that looks up data from another file prior to putting it into a new record in the main file. The plan is the same except that step 4 obtains data and returns with it instead of creating a record. The important thing is that the main file record is created after the 'rlink'.

Reporting Refinements

Printer Features

Superbase includes just one feature for enhancing the printed quality of reports: the underlining command, which can be either '@+' or '@-'. If you want to take advantage of the features offered on most dot-matrix and daisywheel printers, you need to know how to control the printer directly from within Superbase. Fortunately, this is extremely easy.

Almost all print features are obtained by sending a string of control characters to the printer. Often, the first character in the sequence has the ASCII value of 'escape', code number 27; these are called escape sequences. Whatever the sequence of characters required, you can manage it satisfactorily in Superbase, first assembling the code values in a string variable, then using 'print' to send the string straight to the printer. Suppose the control codes for switching on bold face printing were 27 and 14, in that order. Your program lines should look like this:

```
10 p$=chr$(27)+chr$(14)
20 print p$
```

Notice especially the use of the '+' sign to concatenate the strings of the separate codes. This is essential. A line that uses the semicolon instead would almost certainly not work; as Superbase outputs a space after every separate item, the sequence recieved by the printer would include a space in the middle:

```
10 print=chr$(27);chr$(14):rem Equals ESCAPE <space> 14
```

The control string does not have to be sent in a separate statement. It can form part of a regular output line, using

whichever of the commands 'print', 'detail' and 'output' is appropriate:

```
10 output all [name][telephone]p$[balance]n$
```

Here, p$ sends a control string, say to switch on italic printing, and n$ sends another, to switch back to normal printing. The possibilities are limited only by what your printer is capable of doing.

Eliminating the Page Break

Sometimes you may want to print on continuous stationery without the default page break. This is a function of the difference between the values set for 'plen' and 'tlen', which the "start" program takes as 66 and 60 respectively, causing a skip of six lines at the foot of each page.

If you want to eliminate the page break, make the two values the same:

```
10 plen 66:tlen 66
```

Remember to reset them for normal printing.

Printing the Screen: CTRL-P and CTRL-O

The screen dump option is sometimes thought by naive users to be the main way of obtaining printed copies of records. In fact it is intended as an occasional convenience. However, there are circumstances in which the screen dump is valuable, when you want hard copies of your record formats or memo screens or help screens. A simple CTRL-P normally suffices, but you may find if you use the function repeatedly that every third CTRL-P seems to fail mysteriously.

The reason is that Superbase counts the printed lines against the current value for 'tlen', normally 60. As the length of record format and memo screens is 23 lines, two CTRL-P's take the counter to 46, leaving insufficient room for the third screen. The solution is to set 'tlen' to 46, which causes Superbase to start a new page after every two printouts.

The CTRL-O option is not available in all versions of Superbase. It is similar to CTRL-P, except that the top two lines of the screen are also printed. The option is intended to facilitate documentation of the system. Note that you would have to set 'tlen' to 50 to permit repeated printout with CTRL-O.

Page Numbering

The lack of automatic page numbering in the Superbase report generator has sometimes been raised a source of irritation. We

have recently discovered a way of achieving quite satisfactory page numbering using the facilities of the system itself. This is far better than having to resort to counting the lines printed and placing the result in a title subroutine that cannot use the 'title' statement. Here is an example of a routine that prints page numbers:

```
10 report "customers"
20 pn=1:pl=66:h=11:rem h is title + footing lines
25 plen pl:tlen 60:rem 6 footing lines
30 title "CUSTOMER LIST"@60"PAGE:"&2,0pn;cr$cr$
"CUSTOMER"@20"BALANCE"@30"TELEPHONE"cr$
40 total t1=t1+1;x=abs(t1+1+(h*pn)=pn*pl);pn=pn+x
50 detail all [name]&5,2@20[balance]@30[telephone]
60 endreport
```

The key to the program is in line 40, the total line. We take advantage of the 'total' command's ability to count lines automatically, and record the number of lines output with each record in the variable t1. Next, the variable x takes a value each time a record is output, either 0 or 1. This value is worked out as a test of the equality of two expressions. The left hand expression evaluates a figure based on the line count, t1. The right hand side evaluates a figure based on the current page number, pn, times the page length in lines, pl. The last expression on the line increments the page number itself. Whenever x is 0, that is on all lines except the last, pn is incremented by 0, i.e. it is unaffected. But whenever the two are equal, which only occurs at the end of a page, the page number is incremented by one ready for the next page. The actual page number is of course printed as part of the 'title' line. It works!

If you plan to use this routine, be sure you set up the initial variables correctly. The page number, pn, is set to whatever you want, normally 1. The variable h holds the total number of lines that are not printed as part of the body of the text -- the lines printed in 'title', and the difference between the values for 'plen' and 'tlen'. Finally, use a variable for 'plen'; I have used pl, which reappears on line 40 as part of the key expression in the middle.

Retaining Subtotal Values

Although for most purposes the output of a subtotal as part of report directly after the subgroup from which it is derived is adequate, there are times when it is desirable to retain the subtotal for each group, so that at the end of the report all the subtotals can be output as a. summary, or perhaps expressed as a percentage of the whole.

The method for achieving this goal is similar to that for page numbering, in that we make use of the built in features of the report commands. In particular, the 'total' line of the report serves to regulate the general processing. Here is a program that illustrates the techniques involved:

```
10 r$=chr$(13)
14 rem
16 rem *** DIMENSION SU(N) TO MAX GROUPS ***
18 rem
20 dim su(20)
22 rem
24 rem *** REPORT STARTS HERE ***
26 rem
28 report "customers
30 title @30"ARRAY SUBTOTALS"plus
35 r$r$@1"CODE"@14"STATE"@20"BALANCE"
40 total n=abs(s3=0);s2=s2+n;plus
45 s3=s3+[balance];su(s2)=s3;s$=[state]
50 subtotal [state]"Subtotal for state";plus s3
55 s$+":  "@30&8,2s3
60 detail from ""@1[code]@14[state]plus
65 @20&7,2[balance]
70 endreport r$r$
74 rem
76 rem *** DISPLAY RETAINED SUBTOTALS ***
78 rem
80 display "     Number of groups: "&2,0s2;
85 for i=1to s2:
90 display @39"Subtotal for group"+str$(i);
"was:"&7,2su(i)
95 next:wait:menu
```

The first step is to dimension an array with sufficient elements to store a subtotal for each of the groups encountered in the report. You must estimate this in advance, but it does not matter if you overestimate. If you underestimate, your print program will fail after doing most of the printout -- most frustrating, so it might be a good idea to count the number of subgroups in a separate program.

The 'total' line is again the key. In it, there are a number of expressions, some of which are normal report functions, while some are purely for the special purpose of the program. So, 's$=[state]' just assigns the name of the state to a string variable for use in the 'subtotal' line, and 's3=s3+[balance]' functions as a normal accumulator, also printed out on the 'subtotal' line, where it is cleared to zero on each change of subtotal group.

The aim of the other expressions, which work together, is to store the subtotal that accumulates in s3 in the su(n) array, increasing the array element subscript each time the subgroup changes. The variable s2 subscripts the array, and increases by one each time n in the previous expression takes the value 1, which it does only when the subtotal itself is zero -- that is, at the beginning of each new group. In this way, the subscript is always ready for the next subgroup, leaving the amount accumulated in the previous element as the retained subgroup total, thus achieving the aim of the program.

At the end of the program, a separate routine prints out the values of the array, with the final value of the subscript, s2, controlling the end of the loop. The value of s2 also indicates how many subtotal groups there were.

Concatenating Strings for Output

If you want to produce output in which the space that Superbase prints after each element is eliminated, typically when a line of elements separated by commas is required, you must concatenate strings with the '+' sign. However, a line of this form:

```
10 output all across [name]+","[street]+","
```

produces this effect:

```
adams               , 21 Main Street               ,
baker               , 345 North Avenue             ,
etc.
```

The full length of the field is output. The '&' truncation command cannot be used to achieve the desired result. The solution is to set a variable to close up the gap and then print the comma. For screen output the cursor left character works:

```
5 l$=chr$(157)
10 output all across &[name];l$+",";&[street];l$+","
```

For printed output the backspace character should do the job, provided your printer supports it.

More About Key Lists

Empty Lists

Sometimes the result of a 'find' can be a list with no entries in it -- an empty list. This poses special problems. In very early versions of the program attempting to select from such a list actually caused an error. There is a way around this problem, but if you own one of these early Superbases, you need an update! The method is rather cumbersome, but effective. You must first create a dummy list on your disk with one entry in it. Do it with 'find', specifying one unique key from any of your files. When you have a dummy list, the following program will work:

```
10 find "testlist" where [name] is "smith"
20 maintain o "c0:list2=0:dummylist,testlist"
30 for i=1 to 2:select from "list2":eol 50
40 next i
50 if i<3 then display "List was empty":wait:menu
60 select from "testlist"
etc.
```

After 'find' has created "testlist", the program creates a list called "list2" (you must make sure there's no file of this name on the disk before you start) by concatenating "testlist" and "dummylist"; neither of these is affected. If the subsequent test reveals that the 'for ... next' loop only selected once from "list2" before detecting the end of list condition, then "testlist" has no entries in it.

When Superbase was modified to allow the 'eol' test to detect an empty list on the first attempt to select from it, such circumspection became unnecessary; you can now check the list before you go on to process it. However, there is still the problem of what to do if, having established that the list is not empty, you want to start again at its beginning. Before the 'close' command was introduced, many users thought they had to read all the way to the end of the list before they could get back to the first key. Not so. We use one limitation of the program against another. You still need a dummy list, this time one with nothing in it. Then you can set up a test like this:

```
10 find "testlist" where [name] is "smith"
20 e=0:select from "testlist":eol e=1
30 if e=1 then display "List was empty":wait:menu
40 select from "dummylist"
50 select from "testlist"
etc.
```

If "testlist" has no entries, the variable e is set to 1, and line 30 acts accordingly. If "testlist" has an entry, the program forces Superbase, which can only have one list open at once, to close "testlist" by opening "dummylist". Immediately afterwards the same logic allows us to re-open "testlist", starting of course with the first entry, the one we want. The example above assumes that you need to reopen the list from the beginning, and it would of course be possible to carry straight on from line 20 into the main processing after establishing that the list was not empty.

The Apple version of Superbase introduced the 'close' command specifically to make this programming judo unnecessary. Life is much easier:

```
10 find "testlist" where [name] is "smith"
20 select from "testlist":eol display "List was
empty":wait:menu
30 close
40 select from "testlist"
etc.
```

Line 20 will detect an empty list and terminate the program. If the list is not empty, line 30 closes it, allowing line 40 to begin the main processing with the first key on the list.

Finally, the 'count' command, introduced to the Commodore 128 version, provides the perfect solution, eliminating the need for a test read from the list altogether:

```
10 find "testlist" where [name] is "smith"
20 count x: if x=0 then display "List was empty":
wait:menu
30 select from "testlist"
etc.
```

'Count' assigns the number of iterations of the previous command to a numeric variable. It works with these commands:

```
find      batch
sort      output
detail    import
export    select from
```

Adding Selected Keys to a List: Picking

When you are browsing through a file, you may sometimes want to be able to tag the index key of the record you are looking at onto a list to be used in a later operation. I shall refer to this operation as "picking". If you have several records to pick at once, the problem is not so acute. The solution is to set up a special status field for this operation, and put a value into it with 'select r'. Later, a 'find' statement obtains the list in the usual way; 'batch' can be used to reset the status field:

```
10 find "" where [status] is "y"
20 batch from "" [status]=""
```

If there are no convenient criteria for selecting a single record from among the others with 'find', which would be an inefficient method anyway as 'find' always reads the whole file, you need a special program. As a precondition you must set up one of your database files as a working file, one that does not store the usual kind of data records but instead just a single record containing a single field, the index key, which need be only two characters long, with a value of "aa". Once this is set up, a combination of selecting the record from which to pick the key, 'link', and 'output to' can achieve the desired result:

```
20 file "dummy.a":find"h8appd"where [key]is"99"
30 file "customers":setlink "dummy.a":select f:goto 50
40 select n:eof menu
50 select c:ask "confirmation (y/n/x):";op$
60 if op$="x"then menu
70 if op$="n"then 40
80 if op$="y"then 100
90 goto 50:rem op$ was out of range
100 cd$=[code]
110 link
120 output all to "h8appd,a"cd$
130 rlink:goto 40
```

Line 20 creates a list, here called "h8appd", which has nothing in it. We know it has nothing in it because the only record in the file "dummy.a" has a key of "aa", and the 'find' is looking for "99". The reason for the empty list is that later we will want to

append the selected code to it, and you cannot append to a non-existent list.

Next, lines 30 to 90 set up the main file and the link file, and browse through the main file, validating the input option op$ and acting according to its value. If op$ is "x", the program exits. If op$ is "n", the program selects and displays the next record from "customers". If op$ is "y", the key is appended by the next piece of code.

In line 100, the required index key is set into cd$. A 'link' switches to the link file, but we are not interested in any of its record data. Instead, line 120 outputs just the cd$ variable to disk, tagging it to "h8appd". Because there is only one record in "dummy.a", the 'all' part of the statement results in only one iteration -- cd$ is added only once. And because we do not mention a field name, cd$ is the only item that is appended to the list. To finish off, line 130 returns to the main file and resumes the selection and display processing.

The Metacommand: 'do'/'perform'

I call this the metacommand because it functions at a higher level than all the other Superbase commands, which actually serve as parameters for it. I shall be referring to the command as 'do' throughout this section, as most users of Superbase know it by this name. However, if you own a version of Superbase for the Commodore Plus/4 or 128 computers, the name of the command is 'perform'. In the following examples, these users should substitute 'perform' for 'do'.

'Do' has two main uses. It provides a way to process the fields in records on the basis of field names either input through the keyboard or obtained from some other source. And it allows you to construct record formats with highly structured field names and process them in a short loop rather than one by one. The latter use may in fact be generalized to cover all instances where program code can be shortened by employing 'do'.

Let's consider the first use. If you think about it, there is no way you can request the input of a field name and then assign a value to that field, or refer to it at all, using the statements we have looked at so far. This does not work:

```
10 nm$="smith"
20 ask "field name";f$
30 f$="["+f$+"]"
40 f$=nm$
```

No actual field can be referred to with f$. However, 'do' allows you to construct the entire assignment statement of line 40 in a string variable and execute it as such:

```
10 nm$="smith"
20 ask "field name";f$
30 d$="["+f$+"]=nm$"
40 do d$
```

You have to watch carefully how the variable, which I refer to as the 'do' string (maximum length 80 characters), is assembled. Here we have four components:

```
nm$
[
f$
]=
```

All except f$, the name of the field, are enclosed in double quotation marks. Actually, the best way to check the validity of your 'do' statement is to use 'display' as a debugging aid:

```
10 nm$="smith"
20 ask "field name";f$
30 d$="["+f$+"]=nm$"
40 display d$
```

If line 40 displays a valid Superbase statement, you can confidently substitute 'do' for 'display'. In our example, if the user input "name" to line 20, line 40 would produce this display:

```
[name]=nm$
```

The use of literals produces another problem: you cannot put double quotes within double quotes. You must first assign the double quotation mark character to a variable, and then include it in the 'do' string. The ASCII value of the double quotation mark is 34, so line 10 here will assign it to a string:

```
10 q$=chr$(34)
20 d$="display"+q$+"smith"+q$
30 do d$
```

Line 20 assembles a 'do' string that would display as:

```
display "smith"
```

When line 30 is executed, it just displays the content of the literal, "smith".

'Do' can only execute one command at a time -- you cannot have multiple commands separated by colons. However, it is possible to place multiple assignment statements in the 'do' string, by adopting the syntax of the 'calc' command, which permits the use of semicolons. As long as you are referring to field names, Superbase assumes that the 'calc' syntax is intended:

```
10 nm$="Smith":st$="234 Main Street"
20 d$="[name]=nm$;[street]=st$"
30 do d$
```

But if you wish to assign values to variables, you must specify the 'calc' command as part of the 'do' string, even if the data is taken from field names:

```
10 select f
20 d$="calc nm$=[name];st$=[street]"
30 do d$
```

The other main use for 'do' is in assembling structured field names to shorten programs. By structured field names I mean names that include a fixed prefix and a variable sequenced numeric suffix, as in this sort of record format:

```
q1 <    > d1 <              > p1 <        > t1 <        >
q2 <    > d2 <              > p2 <        > t2 <        >
q3 <    > d3 <              > p3 <        > t3 <        >
q4 <    > d4 <              > p4 <        > t4 <        >
```

If we want to program input into these fields, we can use 'do' to avoid a separate 'ask' statement for each one.

```
10 q$=chr$(34)
20 for i=1to 4
30 ln=len(str$(i)):fl$=right$(str$(i),ln-1)
40 for j=1to 3
50 f$=mid$("qdp",j,1)
60 d$="ask"+q$+f$+fl$+q$+"["+f$+fl$+"]"
70 do d$
80 next j:next i:select c:wait
```

There are two loops, one controlling the rows, one to four, and one controlling the columns, q, d, and p. The task is to assemble the field names in the order q1, d1, p1, q2, d2, etc., and request input for them on the command line, each time prompting with the name of the field.

Line 30 obtains the field digit, one to four, from the loop variable i, in a string form in which it can be appended to the field letter. The complex expressions are necessary because the Commodore 'str$' function always precedes the string of the number with a space, which we must eliminate by measuring the length of the string, and truncating with 'right$'.

Line 50 obtains the field letter from a literal string, using the control variable of the inner loop, j, to identify each letter in turn. Line 60 then assembles the 'do' string from the 'ask' statement; the field name,'f$+fl$', surrounded by quotes, q$; and the field name again, surrounded by square brackets. Line 70 executes the 'do' string. If you're uncertain what happens at this point, you can run this program with 'display' instead of 'do' (remove the 'select c' too) without setting up a record format.

PROGRAMMER'S CHALLENGE. *Try to write a program in Superbase or BASIC that identifies a variable and operates on it without explicitly referring to it. This was the problem for which 'do' provided the solution.*

New Files for Old: Reorganizing the Database

The best planned database invariably proves inadequate after you've been using it for a few weeks. When that happens, you probably need to redesign the record formats, and if you have been programming, adjust menus and programs accordingly. But the most serious job is the transfer of your data from one file to another, or even from one database to another. That is what I shall be concentrating on in this section. Much of the discussion also has relevance to the particular problems engendered by the "Disk Full" error, which is covered in the Troubleshooting appendix.

'Export' and 'import'

First we'll look at the easiest case: the extraction of data from one file and the insertion of it into another, where both files are exactly the same in format. For this, you should use 'export' and then 'import'. The correct sequence of steps is as follows:

1. Set up a new file format to receive the data. Select 'other' from the 'maintain' submenu, and type in:

   c0:newformat=0:oldformat <return>

   If you want the new format on a different disk in drive 1 of a dual drive unit, type:

   c1:newformat=0:oldformat <return>

   This results in a identical copy of your file format under a new name. It is not yet part of a database, nor is there any data in it.

2. Select the database in which you want the new file to exist. Select the 'file' option, and type in the name of the new format you created in the last step. Obviously it must be on the same disk. Superbase responds:

   File does not exist: create it?

   You respond "y". Superbase then finds the new format on the disk and makes it part of the database by inserting the name in the 'file' catalog. There are no records in the file yet.

3. Select the database in which the original file exists, and select the file. Now select 'export' from the 'maintain' submenu. To the prompt "Enter export file name", type in the name of a file that can hold the data in sequential format on the disk before you import it into the other file. The name you

type here must not be the name of either the original or the new file format. Superbase now outputs a stream of the data from the original file; this takes a little time.

4. When the 'export' is complete, select the new database and file. Make sure that the export file you created in the last step is on the same disk as the new database if you have a single drive unit. Now select the 'import' option from the 'maintain' submenu. To the prompt "Enter import file name", type in the name of the export file you created in the last step. Superbase now reads the data from the export file and inserts it into the fields of the new format; this continues until all the data has been read. You end up with a brand new file of records in a new database.

CAUTION: *Possible Consequences of 'import' Failure*

Normally, Superbase flushes out its internal buffers whenever you store a record, ensuring that the control information about the database files and indexes is always up to date. However, because 'import' involves the continuous writing of records, flushing is not done until the 'import' operation is complete. If 'import' fails for some reason, often because it tries to insert a blank key field, generating the "Invalid FMS Parameter" error, the database is in a critical state. You must act carefully to avoid loss of data. None of the following operations is allowed:

quit
file
database
restart
power down
exit to Superscript

Any of these could cause the loss of crucial information about the database. If 'import' fails, immediately use 'store' or 'select replace'. Either of these commands will cause the database controls to be properly updated.

'Output to' and 'import'

If the formats are not identical, the situation is more complex. You must either use 'output to' to create a disk file and then 'import' it as above, or, if you want the operation to be wholly automatic, write a special program. 'Output to' is the easiest option. Here's the procedure:

1. Either set up a new format or copy a new format from an old one and select it as in steps 1 and 2 above.

2. If you're working with an existing format, select the 'format' option, and make any modifications you want, such as adding or deleting fields, changing their order or their names, or adding new screens. Whether it's a new or an existing format, add a

dummy text field at the end of the record format. It can have any name, say [dummy], and need be only two characters long. This field is needed to catch the extra return character that the 'fill' statement adds to the data from each record.

3. Select the original file, and then the 'output' option. Type in the following syntax:

   ```
   all fill to "datafile" [field1][field2] <return>
   ```

   The name of "datafile" must be unique. You can output from a list if you want to transfer only a subset of the original file. Your own field names can be in any order and can be a selection from the fields in the format, but must correspond in number and type to the fields in the new format. You must specify any result or calendar fields by name -- don't leave them out or substitute variables. If you have any new fields in the new format, insert a string or a numeric variable at appropriate points in the 'output' line; set the variables to a space and zero respectively first:

   ```
   a$=" ":a=0 <return>
   output all fill to "datafile" [field1]a$a$
   [field2]a <return>
   ```

4. Select the new file, and then the 'import' option, as in step 4 above. If there are any errors, delete any records that have been added to the new file before attempting another 'import'.

5. With the new file selected, use 'format' to remove the [dummy] field from the end of the record format.

File to File Transfer

There is even more flexibility if you use a program to transfer data from one file to another. The basic procedure is to read a record from the old file, assign all the field contents to variables, swap to the new file, assign the variables to the fields of a blank format and store it, and then return to the old file to process the next record:

```
10 file "old
20 select f
30 a$=[code]:b$=[description]:c=[date]:d=[amount]
40 file "new
50 clear:[code]=a$:[description]=b$:
[date]=c:[amount]=d:store
60 file "old":select n:eof menu
70 goto 30
```

Notice that in line 30 the [date] field is assigned to a numeric variable; a string variable here would cause an error. In this method, you do not need to specify any result fields. 'Store' forces Superbase to calculate them before adding the record to the file.

An advantage of this method of transferring data is is that there is no need for the intermediate sequential file used in the methods described above, saving considerable disk space. In fact, if you make the program delete the original record in the "old" file in line 60, that space will be re-used by the next 'store', and the overall size of the database will hardly change at all.

Database to Database Transfer

If you want to transfer data from one database to another, even if they are on different disk units, this program is the one to use. The following example shows how to switch databases, and assumes that the "old" file is on unit 8, the "new" file is on unit 9, and the unit has been set up correctly:

```
10 database "db.one",8,0:file "old
20 select f
30 a$=[code]:b$=[description]:c=[date]:d=[amount]
40 database "db.two",9,0:file "new
50 clear:[code]=a$:[description]=b$:
[date]=c:[amount]=d:store
60 database "db.one",8,0:file "old":select n:eof menu
70 goto 30
```

The program will necessarily run quite slowly, because it has to reselect database and file twice for each record, but it will get the job done.

With further refinement, versions of this program can be used to take data from more than one file and join it together in another, or conversely to distribute data from one file among others, either in the same database or in another one, on the disk unit and drive of your choice. An embarrassment of riches.

Switching from Duplicate to Unique Keys

From time to time, a user discovers that duplicate keys are awkward customers, and decides to switch to unique keys. This is essentially an easy task that makes use of some of the procedures explained above. The new requirement is to add a unique key to the file. This is the recommended method:

1. Set up a new record format as in the previous guidelines. If using 'output to', include a dummy text field. Ensure that the format has a key field long enough for five digits. Do not allow duplicate keys.

2. In the old format, add a text field to the end of the format, five characters long, called [newkey].

3. On the command line, type the following:

```
n=10000 <return>
batch all n=n+1;x$=str$(n);[newkey]=right$(x$,4) <return>
```

This inserts a unique number into [newkey] in each record, in the sequence 0001, 0002, 0003, etc.

4. Use 'output to' together with 'import', or one of the program listings above, to transfer data to the new file, making sure that the fields are assigned correctly, in particular that the [newkey] field is assigned to the key field of the new file.

The new file will hold the records in the same order as the old file, but with unique keys. If you prefer, you can modify the 'batch' statement to construct keys based on an alphabetic element of one of the text fields in the old file.

Miscellaneous Tips

Dates

There are users who need to work out such things as the day of the week from an input date. To them these routines are dedicated. Here is a program that illustrates five different date checking routines. The first part of the program displays a short menu and accepts input of a date in either American or European format. The second part contains the code for the five menu options. Comments and explanations follow.

```
10 rem
20 rem *** SET UP REFERENCE VARIABLES ***
30 rem
40 d3$="SatSunMonTueWedThuFri"
50 d7$="SaturdaySundayMondayTuesdayWednesday
ThursdayFriday"
55 of$="0109152128374551"
60 rem
70 rem *** MENU AND SELECTION ***
80 rem
90 display chr$(147)@10,2"Date Functions
100 display @10,4"1 Get year number"
110 display @10"2 Get day of month
120 display @10"3 Get day of week number
130 display @10"4 Get short day of week
140 display @10"5 Get long day of week
150 display @10,10"Choose
160 wait a:if a<1or a>5then 160
170 display chr$(147)
180 rem
190 rem *** GET DATE, SET VARIABLES ***
200 rem
210 ask "date";dt$:date dt$,n:if n=0then 210
220 [date]=dt$:dt=[date]:convert [date],dt$
230 rem
240 rem *** BRANCH TO OPTION, LOOP TO MENU ***
250 rem
260 on agosub 310,350,410,460,520
```

```
270 wait:goto 90
280 rem
290 rem *** GET YEAR NUMBER ***
300 rem
310 yr$=right$(dt$,2):display yr$:return
320 rem
330 rem *** GET DAY OF MONTH, EITHER FORMAT ***
340 rem
350 x=asc(left$(dt$,1))
355 if x>51 then mn=val(mid$(dt$,4,2)):goto 370
360 mn=val(left$(dt$,2))
370 display &2,0mn;:return
380 rem
390 rem *** GET DAY OF WEEK: SAT=1 ***
400 rem
410 wn=(dt-int(dt/7)*7)+1
420 display &2,0wn;:return
430 rem
440 rem *** GET DAY OF WEEK: ABBRV. ***
450 rem
460 wn=(dt-int(dt/7)*7)+1
470 da$=mid$(d3$,(wn*3)-2,3)
480 display da$:return
490 rem
500 rem *** GET DAY OF WEEK: FULL ***
510 rem
520 wn=(dt-int(dt/7)*7)+1
530 d1=val(mid$(of$,(wn*2-1),2))
540 d2=val(mid$(of$,((wn+1)*2-1),2))
550 da$=mid$(d7$,d1,d2-d1)
560 display da$:return
```

Option 1: Obtaining the Year Number

Line 310. The variable yr$ holds the two digit year. If you want, expand it by prefixing "19": 'yr$="19"+yr$'. Or, convert it into a number with 'yr=val(yr$)'.

Option 2: Obtaining the Day of the Month, Independent of Input Format

Lines 350-370. The code puts the ASCII value of the first character of the date string dt$ into a variable, x. If x is greater than 51, the day of the month is not at the front of dt$, so it must be in the middle. The variable mn holds the day number.

Option 3: Obtaining the Number of the Day of the Week

Lines 410-420. Saturday is taken as day 1, because of the way the expression in line 410 works. It takes the julian date number, and subtracts from it the nearest multiple of seven, adding one to the result to avoid a result of zero for Saturday. The value in wn is the day number of the week. It is used in both the following calculations.

Option 4: Obtaining a Three Character Abbreviation for the Day of the Week

Lines 460-480. The string d3$ holds all the abbreviations for the days of the week. The code first calculates wn as above, then uses it to extract the corresponding three characters from d3$ into da$.

Option 5: Obtaining the Day of the Week String

Lines 520-560. The string d7$ holds the days of the week, and the string of$ holds a series of two character offsets that point to the start of each day in d7$. The code calculates wn as above, then uses it to work out two offsets for d7$, one for the start of the day, the other for the start of the next day. These give both the location of the substring and its length. The variable da$ holds the day.

PROGRAMMER'S CHALLENGE. *Options 3, 4, and 5 can be made noticeably more elegant if you set up some user defined functions to compute the variables wn, d1, and d2.*

A final word about the 'date' statement. You can use it to set the date format as well as to validate dates. I suggest you put it in the "start" program if this is of concern to you:

```
10 date "01JAN84"
or
10 date "JAN0184"
```

Conditional Updating with 'batch'

Although you can use lists for a number of operations, you cannot 'find' from a list. By combining Boolean expressions with 'batch', we can force Superbase to update only those records that fulfil certain criteria. This example uses a list, and updates records where the [balance] field is greater than 1000:

```
100 batch from ""n=abs([balance]>1000);
[amount]=[amount]+1000*n
```

There are some restrictions. You can only use the operators > < >= <= and <>. The equals sign alone is not allowed. Text strings are not allowed, so you can only test numeric fields; and because the effect of the update depends on the value of the numeric variable n in the final expression, only numeric fields may be updated.

You can, however, compare dates in the first expression, though you must put the value of the dates into numeric variables first. This example updates only those records where the date field holds a date in December 1985:

```
50 clear:[date]="30NOV85":d1=[date]
60 [date]="01JAN86":d2=[date]
100 batch all n=abs([date]>d1)*abs([date]<d2);
[amount]=[amount]+1000*n
```

With some refinement, the program could request user input and update the file, eliminating the need for a separate 'find' operation. However, 'batch' used in this way is quite slow, as all the records in the file have to be read and rewritten.

Programming the Command Area

For the completely customized look, you have to program the command area of Superbase, to hide the normal display. This program does it:

```
10 h$=chr$(19):u$=chr$(145):ro$=chr$(18):rf$=chr$(146)
20 b$="Reptile Rentals Inc.
Superbase System"
30 b1$="                          ***** MAIN MENU *****
                        "
40 d$=" "+h$+u$+ro$+b$+b1$+rf$
50 display d$
60 ask @2,2"Enter ";x$
```

The first line sets up variables holding control codes for home, cursor up, reverse field on. and reverse field off. The next two lines set up the strings that will be displayed on the top two lines of the screen. The first should be one shorter than the screen width, the second should equal the screen length. Line 40 creates a single string from a space and the necessary control variables concatenated with the display strings. Line 50 displays it, and then the program can continue as normal. Almost any statement disturbs the top two lines, so you should put the display code into a subroutine and call it as necessary.

Using 'peek' and 'poke'

These two BASIC statements are useful for such things as changing the screen colours directly, or switching off automatic repeat on the keyboard. They work by looking at and changing the contents of memory, and if used carelessly could in theory cause Superbase to damage your database. For this reason, they are limited to the command line; you cannot use them in a program.

Duplicate Keys

While I have emphasized the desirability of avoiding duplicate index keys, there are many users who for various reasons prefer to keep them. These users cannot make proper use of the 'find' option unless they devise another way of reading from a list than the normal closed iteration of commands like 'output' and 'detail'. The following program shows a way of using a link file holding the

index keys from a 'find' as a way of reading records correctly from a main file:

```
10 elink
30 file "index":setlink "data"
80 select f
84 rem
86 rem *** LINK ON INDEX FILE RECORD KEY ***
88 rem
90 k$=[key]
100 link k$:nmat 180
110 pmat 180
120 rem
122 rem *** CHECK CRITERIA AND PROCESS ***
124 rem
130 if [code]=k$ and [amount]>2000then 150
140 goto 180
150 display [code][description][amount][result]
152 rem
154 rem *** GET NEXT DATA FILE RECORD ***
156 rem
160 select n:eof 180
170 goto 130
172 rem
174 rem *** RETURN FROM LINK ***
176 rem
180 rlink
182 rem
184 rem *** GET NEXT INDEX KEY FILE RECORD ***
186 rem
190 select n:eof 220
200 if [key]=k$then 190
210 goto 90
212 rem
214 rem *** END OF INDEX KEY FILE ***
216 rem
220 display "End of report":wait:menu
```

An index key list can be imported into a special one field format (with duplicate keys allowed in this case), but this will generate the "Invalid FMS Parameter" error at the end of the import process. This is caused by the extra return that Superbase appends to the end of a key list. You should immediately execute a 'store' or a 'select r' on the file to avoid possible loss of data (see above, on possible consequences of 'import' failure'). Despite the error, the above program should work satisfactorily.

The program links on the first key from the index file. If it finds a matching record in the data file, it checks to see whether the record fulfils the same criteria as were used in the original 'find'. This is essential. If it does, record details are output -- any processing could be substituted here. Then the program, still in the data file, reads the next record. This is how it accesses records with duplicate keys, which are only available to a sequential 'select n'. The process of comparison, processing, and reading is repeated until a key that does not match the 'link'

key is found. The program then returns to the index file. There, it reads through any further records with the same key; their details will already have been output. When it finds a different key, it returns to line 90 to assign it to k$ for linking. The end of the program is determined by the index key file.

CHAPTER 11
PARAMETERIZING THE SYSTEM

The best starting point for the concepts discussed in this short chapter is the Superbase "start" program, which we have already encountered in connexion with menu programs. "Start" gives you control over the initial conditions in which your Superbase system functions. You can extend this idea considerably. With care, you can set up one or more files that exist specifically to enhance the flexibility of your system. More than one type of file can be used. You can have both database files with file formats dedicated to system parameters, and library files created with 'memo' to hold sets of variables in textual form. On the output side, you can create sequential files from Superbase functions such as 'directory' and 'status', as well as the more usual data files. A survey of the attributes and purposes of these types of file follows.

Database Files

The file is created in exactly the same way as any other database file format, with the commands 'file' and 'format'. The information held in the file varies according to the type of system, but some typical fields might be held in this form:

```
SYSTEM FILE                                                Screen 1
-------------------------------------------------------------------
Code <    >
---------------------- Company Details --------------------------
Name      <                                          >
Street    <                                          >
City      <                                          >
State     <         >    Zip <          >
Tel.      <                 >
Telex     <         >
---------------------- System Details ---------------------------
Date last used  <          >      Last backup <           >
Last user ID    <          >      Last archive <          >
-------------------- Transaction Numbers ----------------------
Invoice   <      >     Inventory  <      >     Cash <      >
-------------------- File Update Record -----------------------
Customers <   >        Invoices   <   >     Inventory  <   >
Cash      <   >        Personnel  <   >     Statistics <   >
```

When the system is first used, company details are entered here. The advantage of such a system is that any report program which needs to print such details does not have to hold them as part of itself; it just reads them from this record and prints them. This principle can save a noticeable amount of memory if adopted for all report programs.

Keeping track of the date allows your system to prompt for certain kinds of action; for example a month end set of updating and reporting routines could be run automatically. If you kept more extensive date information, the system could remind you of appointments, approaching deadlines, or the interval since you last went to the dentist or saw the bank manager.

Taking backups regularly should become a habit for all except those whose data has no value to them. If you store details of backups you stand more chance of keeping to a system and avoiding expensive and frustrating failures.

Transaction numbers can be held here, read into variables at the start of processing, and then written back to this record at the end of it. During processing, each time a new record is created the transaction number is incremented. This automates a standard accounting procedure, and if things go wrong it's easy to edit the record and reset the transaction numbers. The same principle can be used to keep track of the last key added, very useful if your index keys follow a predictable sequence that the data entry program can use to generate the next key.

The file update record fields are updated every time your system detects that a file has been changed or had a record added to it. A field would hold the value "yes" if its file had been updated. This data can be used in conjunction with the backup details to ensure that backups are done only when necessary.

By extension, a record format like the one above could be used for all kinds of data storage specific to the application. You could store flags to show that a particular file needed updating -- more economical than a flag in each record of the file. Fields could be used to store figures for the maximum number of records permitted in a file; the system could then warn you of an approaching "Disk Full" condition.

Sequential Files: 'dump' and 'set'

These two Superbase commands can be extremely useful. Respectively, they save and load plain sequential files from the disk.

'Dump'. When you're programming, typing 'dump <return>' on the command line gives you an instant look at the values of all the variables in your file, except for array data. This can be quite an aid to debugging.

If you type 'dump "filename" <return>', the results are much the same, but are placed in a disk file. The variables are listed one per line. This provides an alternative method of storing parameters for your system to use later on. If you only want to dump some of the variables in the system, make sure that the ones you don't want to dump are held in array elements. A further function is to preserve variables from one program for use by another.

You can write specified variables to a disk file, by using the technique that I described in the section on "picking" records. You set up a database file with only one record in it, consisting of a single key field, contents immaterial. Then you assemble the strings you want in the program that is to do the outputting:

```
10 c$=chr$(34): rem Double quotation marks
20 a1$="a$="+c$+"+a$:a2$="b$="+c$+b$
```

If a$ had the value "John" and b$ had the value "Smith", variables a1$ and a2$ would have the respective values:

```
a$="John
b$="Smith
```

This will allow a future 'set' command to read them successfully. The remainder of the output routine selects the dummy file and executes an 'output' command:

```
30 output all down to "dumpfile" a1$ a2$
```

The line could of course be extended with 'plus', so there is no practical limit to the number of variables that can be manipulated in this way. Remember that if you prefix the dump file appropriately, you can view it with 'help'.

'Set'. The converse of 'dump', 'set' is always followed by a file name. It reads the data from the file line by line, assigning it to the specified variables. A typical 'set' file, which might have been created by 'dump' or 'memo' or by other more esoteric means, looks like this:

```
dt$="01JAN86"
tn$="000456"
e=1
ar(1)=34.23
ar(2)=890.50
```

Note that although you can 'set' data into an array by this means, you cannot write it out again with 'dump'.

One advantage of using 'set' in a program is that it allows you to set up variables or execute some Superbase commands such as 'display' without explicitly including them in the code of the main program. Superbase executes each line of the 'set' file in turn (internally using the 'do/perform' statement), as it reads it from the disk. This keeps memory usage down, but the penalty is that execution of the statements in a 'set' file is very slow, especially when a 1541 disk drive is being used. The other disadvantage lies in the temptation to the programmer not to document the variable assignment. It can be quite mystifying to look at code that suddenly starts using variables that do not seem to have been assigned values.

Sequential Files for System Output

The Commodore 128 version of Superbase includes a command 'open'. This is invaluable for creating disk files from the various sources of output in the Superbase system. It works like this:

1. On the command line, type:

```
open "filename" <return>
```

2. Now select just about any Superbase operation that produces output, either for the screen or the printer. When you execute it, the output will go to the named disk file rather than the normal output destination. Options include:

```
output
detail
print
display
status
directory
catalog
list
```

3. When the operation is complete, type 'close'.

The resulting data file can be manipulated like any other. It can be edited with 'memo', viewed with 'help', read by a word processor such as Superscript, or even imported into a Superbase database file (in some cases you might have to preprocess the file with 'memo' or Superscript to eliminate blank lines)

A final word. Although the above procedure is for C128 Superbase users, you can in fact perform a similar operation with Superbase on the Commodore 64, although we must emphasize that this is a convenience rather than an official feature of the program. Instead of 'open' and 'close', you use the 'pdev' command to reassign the output device identity from the printer to the disk unit:

```
10 pdev 8,8,8
20 print "program listing"
30 list
40 pdev 0
```

This program would list the current Superbase program to a disk file. Line 20 opens the file, 30 executes the command -- you would substitute one of the commands in step 2 above at this point -- and line 40 is the essential equivalent of 'close', which must follow before any other kind of output is attempted (my use of the parameter 0 for 'pdev' here is only an example; you would put in your normal parameters, usually those set in the "start" program). You must use 'pdev' to close the file immediately if the operation fails, or you could be left with an open file on the disk.

APPENDIX

TROUBLESHOOTING

APPENDIX
TROUBLESHOOTING

Nobody's perfect. However careful you are out there, things go wrong. Sometimes this is due to your not understanding the program, but it can have physical causes, and there have indeed been a few bugs in Superbase over the years. In this appendix, descriptions of the commonest causes of users' problems are followed by a detailed listing of all the known bugs in all versions of Superbase.

We strongly recommend that you obtain the latest version of Superbase, which has had all these bugs removed, and comes with a useful utility program which allows you to copy and usually recover damaged databases. Superbase version 2 uses disk space more efficiently, too.

To obtain a new disk, registered users should contact, in the UK:

Precision Software Ltd., 6 Park Terrace, Worcester Park, Surrey KT4 7JZ. Telephone 01-330 7166. Telex 8955021 PRECIS G.

In the USA:

Progressive Peripherals and Software, 464 Kalamath Street, Denver, CO 80204. Telephone (303) 825 4144.

Physical Errors

Superbase will not Load

The program disk may be faulty. Try the backup copy. If both fail, suspect another cause.

On the C128, ensure that the 40/80 key is down if you are using RGBI 80 column output, up if using the video output.

Move the monitor or TV well away from the disk drive. Frequency emissions can interfere with loading. They can also come from the local power supply or electrical equipment.

The disk drive read/write heads may need re-aligning, even if the problem does not show up with all your programs.

Ensure all equipment is switched on if connected, and connected properly.

Some printers can interfere with the I/O chip. Problems have been experienced with the 1515, 1526 and DPS 1101. The 1515 needs a ROM upgrade, the 1526 needs a revision 7 upgrade from Commodore. To test for this problem, try loading with the printer disconnected.

Interfaces, especially serial ones, can interfere with the I/O chip. Non-compatible IEEE interfaces may also cause trouble. Precision Software recommend the Brain Boxes brand of interfaces.

Cartridges, changes to ROM, or any software or hardware item that affects device addressing may prevent Superbase from loading properly.

The computer may be faulty! It happens.

Persistent Read/write Errors

The disk unit may need servicing: the heads may need aligning, the I/O chip could be faulty, or the power supply might be failing.

The disk surface may be contaminated, either by a substance or through exposure to a powerful magnetic source such as a telephone (don't put the phone on the disk, ever). Poor quality (cheap) disks fail more frequently than better quality ones.

Printer Errors and/or Garbage Printout

The 'pdef' and 'pdev' commands must be correctly executed for the type of printer.

If your interface converts output characters to suit your printer, Superbase's output may not arrive at the printer in the same form in which Superbase sends it. As a general rule, Superbase needs only the simplest of interfaces; for example, for a standard Centronics parallel printer connected to a Commodore 64 or 128, a cable from the user port to the printer is all that's needed (set 'pdev 0').

Apple users please note that Superbase supports only the Apple Parallel card and the Apple Super Serial card. Other cards may work, but unpredictably.

PET / 700 / B128 users requiring interfaces that convert RS232 to IEEE or Centronics to IEEE. In the UK, recommended interfaces are Small Systems Engineering B300, Aculab, or IBEC.

See also Output errors.

Disk Errors

Disk Full

This can be quite serious, and can result in a data or index mismatch if the error occurred during a write to a database file. Mismatches are discussed below.

After the disk full error, the simplest solution is to start using the backup disk as the master. Create space to ensure the disk full error does not recur. Start work with a new disk as soon as possible. Also, take a backup of the backup before you start work.

In no circumstances should you do any kind of write operation on a disk that has produced the disk full error.

If you have Superbase version 2, use the "utility" program on the Superbase disk to recover and copy the database to a new disk. This also compresses it. You may lose the data that was being written when the disk full error occurred, unless you were already using version 2.

If you have Superbase version 1, life is not so easy. If there is any space on the disk after deleting lists, help files, etc., use the 'output to' command to extract record data a little at a time using a series of key lists created with 'find'. For example, put '=a*' as the criterion for the key field to extract all records whose keys begin with "a". After using 'output to', copy the data file to another disk with a non-Superbase utility. (If you have a dual drive -- but not twin units -- both the key lists and the data files may be created directly on the other drive.) Copy the file definition(s) across too. Recreate the database, name the database files, modify each one with a dummy text field at the end, and import the data. See the section on Database Reorganization (Chapter 10) for further details.

Purchase of Superbase version 2 is strongly recommended.

Commodore DOS Error Messages

Refer to your Commodore Disk Drive Manual. Superbase does not create these errors, it just reports them.

Database, File and Record Errors

Caution is necessary as an error of this type may manifest during an operation which did not cause it.

Data or Index Mismatch

An error of this type is caused by a discrepancy between actual and predicted block control information.

Copy and repair the database with the Superbase version 2 "utility" program if possible. If not, follow the method described above to transfer data to a new disk. See Database Reorganization in Chapter 10 for further details.

If 'output to' fails because it cannot read a record in the file, and there is a significant amount of record data after the record with the error in it, you may need to write a program to store the data in another file from which it can be output successfully.

1. Use 'batch' to count from the start of the file to the point at which the error occurred.

2. Then use the count to control the end of a loop that starts at the beginning of the file, reads a record, assigns the field data to variables, switches to another file and stores the data, and then returns to the original file and deletes the source record before processing the next record (remember, 'select n' is not necessary after 'select d').

3. Write a similar program to start at the last record in the file and read backwards to the point at which the error occurred, storing the data in a new file.

"*File Definition Invalid*" *when Selecting File*

Often caused by having used the name of a database file as the name of a key list, data file, memo, or word processing file. Rename or delete the offending file. Reselect the database. Then use 'file' to select the file. When prompted to create it, respond "y". Redraw the format as it was before, ensuring that the order and types of the fields are correct; the position on the screen does not matter. After completing the 'format' you should be able to access the records without difficulty. See Chapter 3 for explanation of file definitions in relation to the database itself.

"*No Fields Defined*" *when Accessing File*

Caused by quitting from 'format' before defining any fields or after erasing all existing fields. First, execute 'database' and then 'file' to try and reset the file definition. If you still get an error, then scratch the file definition, and create the one you want under another name. Copy it to the original name, then select 'database' followed by 'file'.

"*Syntax Error*" *when Selecting a Record*

This error can be perplexing. The system stops for no apparent reason when reading a file, and gives this error. You didn't do anything! The error is caused by an earlier action which Superbase cannot detect.

The usual cause is an illegal character in the record data, most commonly the double quotation mark, chr$(34). Use these commands:

```
a$=chr$(34)
find "" where [field1] is a$+"-"
```

[Field1] stands for the first field in your format. This creates a list, probably with just one key in it.

Dispose of the error with this procedure:

1. Select the record with the error. Even though you get the "Syntax Error" message, this is still the current record.

2. Use 'display' to show the contents of each numeric or date field in turn. The bad one will produce the same error message.

3. Use 'calc' to set the offending field to zero.

4. If the unwanted character is in a text field, you should be able to use 'select r' to edit it out.

5. Store the record.

Syntax errors can also be caused by modification of the record format. If you insert or delete a field at any point other than the end of the format, you may cause Superbase to try to assign invalid data to a date or numeric field. Solution: change all field types to text and review the contents. Make the necessary changes.

"*Formula Too Complex*"

This can occur in 'format' or in program execution. The usual cause is too many sets of parentheses in a result field calculation or a BASIC expression.

Illegal 'do/perform' strings can also produce this error.

"*Invalid Calculation : Re-enter*"

Occurs during 'format' while entering a result field calculation. Causes can be the wrong number of parentheses, or an invalid field name, function or operator.

"*Screen Deselected*"

Applicable to C128 only. In 80 column mode, selection of a 40 column file format from 'file', or the use of the 'mode' command, disables the 80 column screen, and shows the above message.

Similar action starting from 40 column mode disables the 40 column screen, but no message is shown.

See the Superbase Manual, Appendix G.

Failure to Achieve Duplicate Keys

Caused by over-eager key pressing and keyboard buffer not clearing. Solution: stick rigidly to this sequence at the end of 'format':

f1 <RUN/STOP> <y>

or CTRL-C for Apple versions

Solve existing problems by re-entering 'format' and terminating it carefully.

Output and Other Record Processing Errors

Printer Features Inaccessible

Generally caused by code type confusion where the printer requires standard as opposed to Commodore ASCII codes.

Superbase works internally in Commodore ASCII. If your printer requires normal ASCII codes, and you have set 'pdef' accordingly, then Superbase converts all printable characters to ASCII before sending them. This can cause a problem if you are attempting to send a specific control sequence.

For example, if you want to send ESC E 65, as 'chr$(27)+"E"+chr$(65)', the "E" will be converted internally to its true ASCII value, which is what the printer requires. However, Superbase also converts the character 65, as this is a lower case "a" in Commodore format, and it emerges as 97 -- the true ASCII value of "a". The control sequence is thus invalid.

Unfortunately, if you set 'pdef' to output Commodore ASCII codes, the sequence still comes out wrong because the value of "E" is not what the printer requires.

Solutions: If your printer requires ASCII codes, set the correct 'pdef' value for normal printing. When you need to send a control sequence, first reset 'pdef 0' to switch to Commodore ASCII. Then send the sequence expressed entirely as character strings, using true ASCII values:

```
chr$(27)+chr$(69)+chr$(65)
```

Superbase sends the codes for the printer without converting them. When the control sequence has been sent, switch back to your normal 'pdef'.

Alternatively, leave the code set for true ASCII. Determine the actual ASCII character required, i.e. 65 is in fact "A", and send that instead:

```
chr$(27)+"EA"
```

This allows you to send features in the middle of an output line.

Overflow

If a number is too big for its output format, either in a record field or in output controlled with '&x,y', hash symbols are displayed:

```
######.##
```

Extend the field format or change the output format.

Fields Appear in Wrong Column

Allow for the extra space after each item. Allow for defaults -- the full length of a text field as defined in the record format, 14 places for a numeric field (9,2 plus one for sign, one for decimal point, and one at the end). Use formatting commands (see Chapter 4).

If the margins are set too close together, output is disrupted.

Fields Appear on Wrong Line

This is most often caused by trying to print at a column position already covered by output data earlier in the line. See previous item on default field lengths. Unformatted numeric fields require 10 positions to the left of the decimal point. As above, check the margins.

Unwanted Blank Line

Check the 'space' and 'lfeed' commands. A blank line can be caused by outputting a character at the end of a line, in column 39/40 or 79/80 (depending on screen width). A semicolon after the final character should eliminate the problem.

Wrong Field Appears

You used the wrong field name. See Chapter 3.

Records Appear in Wrong Order

This is due to a problem inherent in the use of duplicate keys. Switch to unique keys. See Chapter 10.

Total Failure to Output or Process

Is the printer on line?

May be caused by illegal characters in the record or a file corruption.

'Find' Does Not Work

If the 'find' operation hangs up or puts no records in the list

when you know there should be some, there are various possible causes:

Reserved search criteria characters may be in the record data. Solution: remove such characters, pick keys one by one (see Chapter 10).

The disk may be full.

'Sort' Does Not Work

'Sort' cannot work properly with duplicate keys: each attempt to access a duplicate key only accesses the first key in the sequence.

To sort on a text field containing numbers, ensure that all entries are of the same length, padded if necessary with leading zeroes.

Illegal characters in the fields being compared can cause failure.

A full disk prevents Superbase from creating the intermediate or final sort files.

"Out of Memory" on Import

This error can occur if Superbase tries to import strings longer than 255 characters. Such strings can be created inadvertently by 'output across to', which does not insert carriage return characters between fields. Data exported from a non-Superbase system could cause this error.

"Invalid FMS Parameter" on Import

Commonest cause is trying to put a carriage return only into a key field. Ensure that the structure of the record format corresponds to the structure of the fields in the import file. If you rename the import file to start with "h" or "h8" as appropriate, you can look at its contents with 'help'.

Illegal characters may also cause this error.

Loss of Data Following Import Error

If you import records and then the import fails, you will probably still be able to see record data in the file into which you were importing. But if you quit from Superbase, you may find in the next session that the data has disappeared. This is because an error during 'import' prevents Superbase from writing essential control information to disk. At the start of a new session, the lack of this information makes it impossible for Superbase to locate the record data. Always do a 'store' or a 'select r' before quitting if you've had this error. See Chapter 10.

Miscellaneous

No Help

The 'help' screens are stored on the Superbase program disk. When you create a training disk, Superbase copies them onto it. If you want the 'help' screens on another disk, you must either copy them onto it yourself, or use the Training Disk Creation option on the start up screen, then remove items you don't want.

When entering the name of a 'help' screen, or a list or other file you want to view with help, omit the "h" (or "h8" or "h4" if appropriate). 'Help' supplies the correct prefix automatically.

No "labels" or "delete" Programs

These programs are supplied on the Superbase disk, and should be transferred to your own disk before use.

The Superbase 64 version 1 and Superbase Apple version is called "labels".

Superbase version 2 and Superbase 128 have two programs, one called "labels", the other "makelabels". "Delete" is only available on these versions.

Use 'load' to load the program from the Superbase disk. Switch disks, and 'save' the program under the same name onto your work disk.

Guide to Known Problems

Ask &>128 fails
Version/Machine: 2.02 / 64:96:700/B128:264:AII

Ask for string >128 characters -- loses first character.

Ask allowed too much edit
Version/Machine: 1.0C / 64:96:700/B128

During Ask command could edit out of ask field length.

Ask fails with negative number
Version/Machine: 1.0B / 64:96:700/B128

Ask [number] would not accept a negative input.

Ask for field wrong result
Version/Machine: 1.0B / 64:96:700/B128

Ask [field] when record invalid, i.e. EOF, gives wrong result.

Record has failed to clear but is invalid due to condition such as EOF. Entering field value causes results to be evaluated incorrectly.

Ask length error
Version/Machine: 1.07 / 64

Ask not setting required length sometimes allows nulls.

Backup dual drive
Version/Machine: 1.0E / 64:96:700/B128

Closed database after dual drive backup. Had to execute database command again.

Backup single drive 8096
Version/Machine: 1.0N / 96

Try to execute single drive backup -- crashed machine.

Backup crashed after reading 113 blocks of the source disk. No 4040 versions released.

Batch clears screen
Version/Machine: 1.0N / 64:96:700/B128

Executing a batch command cleared the screen.

Calc failure
Version/Machine: 1.08 / 64

Calc failed to execute after first parameter.

calc [a]=x;[b]=y Following parameters would not get executed if a preceding field was involved in a result that did not have all its parameters set.

Calc from menu
Version/Machine: 1.06 / 64

Does not prompt enter calculation displays numbers.

Only occurs if current record invalid

Calc when blank fields
Version/Machine: 1.0E / 64:96:700/B128

Calc where blank fields now clearing properly.

Calendar field >11 characters
Version/Machine: 2.02 / et al.

If calendar field >11 characters then got rubbish.

Cannot set a result
Version/Machine: 1.0a / 64:96:700/B128

Certain fields not accepted in a result formula.

Fields 32,34,39,40,41,42,44,46 could not be involved in a result.

Centronics Printer Problems
Version/Machine: 1.09 / 64

Could not set margins, tlen or plen properly on Centronics printers.

Margins got reset to 40 columns. Used screen size as page size on Centronics.

Command lines >120 characters
Version/Machine: 1.0E / 64:96:700/B128

System would not allow command lines > 120 characters.

Conditional to invalid line
Version/Machine: 1.0L / 64:96:700/B128

e.g. EOF 20 where line 20 does not exist -- went to next line.

Cursor out of range
Version/Machine: 1.0D / 64:96:700/B128

Could send cursor out of range with ask command etc.

Data length in excess
Version/Machine: 1.0L / 64:96:700/B128

If data particular length then stored 256 bytes too much.

If null at end of data falls on page boundary then fails. Can cause system crash when retrieving record.

Data length in excess 2
Version/Machine: 1.00 / 64:96:700/B128

If reading record created by previous bug crashed machine.

If null at end of data falls on page boundary then fails. Can cause system crash when retrieving record --- now fixed.

Data mismatch duplicate keys
Version/Machine: 1.0D / 64:96:700/B128

Entering many records with duplicate key caused data mismatch.

Occurs if enough duplicate keys to fill index block.

Data mismatch/Delete error
Version/Machine: 1.00 / 64:96:700/B128

If record stored twice when not current record caused error.

Record is stored and immediately stored again usually under program control. The second store will replace the current record. Storing a record does not make it current, so could replace wrong one.

Database command
Version/Machine: 1.03 / 64

Database did not clear all file definitions or current file.

Could give mismatch errors if disk swap then database command without file command.

Database names only 16 characters
Version/Machine: 2.02 / AIIdos3.3

Should allow names up to 30 characters.

Date entry/display failure
Version/Machine: 1.04 / 64

Date entry or conversion caused return to menu. Only occurred after dump command executed.

Date field on bottom line
Version/Machine: 1.0L / 64:96:700/B128

Could not set date on bottom line of screen in format.

Date field on bottom line
Version/Machine: 1.0N / 64:96:700/B128

Date set on bottom line in last column -- wraps screen.

If you set a date on the last column the end marker appeared on the top of the screen. Could not erase it.

Del in program editor
Version/Machine: 2.02 / AII

Del command not working properly.

Delete file index mismatch
Version/Machine: 1.05 / 64

Deleting last record gives index mismatch.

Delete record crash
Version/Machine: 1.05 / 64

Delete a record crashes machine, leaving data mismatch.

Deleted space problem
Version/Machine: 2.02 / 64:96:700/B128:264:AII

Deleted space not re-used for index blocks.

Detail only failure
Version/Machine: 1.0E / 64:96:700/B128

Could not use detail command for current record.

Detail should be able to just send the detail from the record selected if all or from are not used but it didn't work properly.

Disk full failure
Version/Machine: 2.02 / 64:96:700/B128:264:AII

Disk full not handled correctly.

If disk got full during index update then could not complete storing data in the index : system left in indeterminate state.

Display wrong position
Version/Machine: 1.0C / 64

Line wrap on screen causing display position to be wrong. Occurs if data printed longer than screen width.

Do string= failure
Version/Machine: 1.07 / 64

Do would not set a string variable properly.

do "a$= ... fails to set string correctly.

Dump null strings garbage
Version/Machine: 1.07 / 64

Dumping null strings i.e. a$= gave rubbish on screen.

Dump second page garbage
Version/Machine: 1.09 / 64

Second page of dump would give garbage.

Dump to device <>8
Version/Machine: 1.08 / 64

Could only dump to device number 8.

Did not allow any disk device <> 8.

Duplicate key problems
Version/Machine: 1.0E / 64:96:700/B128

Index failure when many duplicate keys inserted & deleted.

End field last line/column
Version/Machine: 2.02 / AII

Did not handle a field which ended on the last column of the last line.

End of page unexpected
Version/Machine: 1.07 / 64

If display @x,y where y=current line or lmarg >0 failed.

If column not zero or lmarg not zero then display at a particular line failed and gave end of page message.

Enter can edit form
Version/Machine: 1.07 / 64

If key exists in enter can then edit form around field. After error mode set incorrectly allowing screen edit.

Enter cursor left failure
Version/Machine: 1.0E / 64:96:700/B128

If long field cursor left fails if after 127th position.

Enter insert on last character
Version/Machine: 1.0C / 64:96:700/B128

In Enter if on last character of field could do insert.

Insert at this position corrupted screen layout. Should have been disallowed.

Enter Key exist hang/clear
Version/Machine: 1.0J / 64:96:700/B128

If enter on screen <>0 and key exist could clear or hang.

Export failure
Version/Machine: 1.05 / 64

If down set then export giving extra returns.

Could not import an export file.

Export failure data mismatch
Version/Machine: 1.03 / 64

Export stopped if data mismatch encountered.

Data mismatch now made soft error to export command. Clears buffer and continues exporting.

Export failure fms parameter
Version/Machine: 1.05 / 64

Export stopped if fms parameter error encountered.

FMS error now made soft error to export command. Clears buffer and continues exporting.

Export from "name" "name"
Version/Machine: 2.02 / 64:96:700/B128:264

Added syntax export from "name" "name".

Field name or brackets error
Version/Machine: 1.07 / 64

If using 12 character field names got error incorrectly.

Also gave invalid line re-enter if in program edit mode.

Field names can't be used
Version/Machine: 1.07 / 64

Typing a valid field name gives an error.

Only happens if SORT used then return pressed to exit to menu.

File definition corrupting
Version/Machine: 1.07 / 64

Losing part fields on entry, colours wrong,screen funny.

Errors usually attributable to parameters for another file in a multi-file system being applied to current file after 4th file command.

File Definition Invalid
Version/Machine: 1.0E / 64:96:700/B128

Could overwrite file def by finding a list with same name.

File names should not have been used for lists.

File fails to create
Version/Machine: 1.0J / 64:96:700/B128

System fails to put user in format if file does not exist.

Do File "xx", Create it? "n" then do file with same name did not go to format. On 64 screen goes all same colour.

File manager problem duplicate keys
Version/Machine: 2.02 / 64:96:700/B128:264

Enter duplicate key, esc q, select c, select r.

Screwed up file somehow as select c did not reselect record.

File manager problems
Version/Machine: 1.0P / 64:96:700/B128

1/Many duplicate keys 2/Many deletes and adds.

1/ If many duplicates added can cause index split to fail. 2/ After many deletes and adds can cause index to become unsorted if problem left then delete errors can occur.

Filename problem
Version/Machine: 1.07 / 64

Cannot use filenames >16 characters i.e. 1:abcdefghijklmnop.

FMS Parameter Error
Version/Machine: 1.09 / 64

Using a key list with blank lines in it gives error.

Blank line taken as a key with length 0 this is an error in FMS.

Format allowed record >1108
Version/Machine: 1.07 / 64

Format not recognizing record too long.

System failed if record stored in format too large. Crash on storing/selecting record.

Format Crash
Version/Machine: 1.07 / 64

Too many comments in format crashed system.

Garbage displayed etc.

Format crash record >1108
Version/Machine: 1.0G / 64

Format not recognizing record too long if new file.

Only if a new file -- existing file extensions are ok.

Format field at bottom right
Version/Machine: 1.0L / 64:96:700/2128

Format now allows fields terminating at bottom right corner.

Format loses comments
Version/Machine: 1.0H / 64:96:700/B128

Record too long in format caused system to lose comments.

Format losing numeric format
Version/Machine: 1.07 / 64

Numerics empty during re-format or 1 character at enter.

Format or file error
Version/Machine: 1.0K / 64:700/B128

Screen roll on B128/700/B128 program loss on 64 on certain files.

Format when file exists
Version/Machine: 1.03 / 64

File command on existing file went into format.

Unexpected entry into format option. This problem occurred when using a multi-file system. Did not recognize file as existing.

Function key cause crash
Version/Machine: 1.0K / 700/B128

If function key defined > 10 characters with no space in front. Causes system crash.

Garbage on screen
Version/Machine: 1.07 / 64

Garbage on screen if list stopped with quotes mode on.

If list stopped with uneven number of quotes on screen.

Get command added
Version/Machine: 1.0C / 64:96:700/B128

Get command allowed on this version onward.

Graphics in fieldnames
Version/Machine: 1.0B / 64:96:700/B128

If graphic in front of fieldname then taken as part of name.

Undesirable to have graphics in names. Users don't always realize why they cannot refer to the name.

Graphics in format Apple IIc
Version/Machine:2.02 / AIIc

Inverse video in format changed to graphics.

When changing screens inverse upper case changed to graphics.

Help/Memo in program
Version/Machine 1.09 / 64

Help or memo command using string variable failed.

Help a$ did not find correct file.

If then failure in direct mode
Version/Machine: 1.0C / 64:96:700/B128

If then command from status line invalid but not trapped. Executed program in memory from unknown line number.

Illegal quantity error
Version/Machine: 1.09 / 64

Unexpected error -- will not repeat if line reexecuted.

Error happens after record stored.

Import gave Out of Memory
Version/Machine: 1.0L / 64:96:700/B128

If no file definition in memory import tries then gives out of memory error.

Import textual date allowed
Version/Machine: 1.0E / 64:96:700/B128

If date is in text form could not import it.

Index mismatch reading
Version/Machine: 1.02 / 64

Get index mismatch error on reading record.

Index block has got to 255 characters long. File manager cannot then read the index block. Data is ok.

Invalid Numeric Result
Version/Machine: 1.0R / 64:96:700/B128

Calculation gives invalid numeric result or rounding error.

If field format 6,2 or 7,2 then calculation like 12.01-12 gives a result in BASIC like 9.99999997e-03 as rounding is not done by Superbase.

Invalid parameter error
Version/Machine: 1.09 / 64

Unexpected error will not repeat if line reexecuted.

Error happens after record stored.

Invalid screen number
Version/Machine: 1.07 / 64

Invalid screen number error not stopping program.

Program continues after error encountered.

Key > 30 characters
Version/Machine: 1.0L / 64:96:700/B128

Key field could be made > 30 characters by insert in format.

FMS allowed you to store keys > 30 but did not retrieve record correctly. Overwrote part of record.

Key exists invalid
Version/Machine: 1.05 / 64

Key exists error when key does not exist.

Can occur if record deleted then re-entered.

Key field corrupted
Version/Machine: 1.0L / 64:96:700/B128

Key field appears corrupted if multi-file system.

When key fields are in different positions in two files the system does not swap the positions with the file.

Link working slowly
Version/Machine:, 1.07 / 64

Link loading file definition when already loaded.

List with no keys
Version/Machine: 1.0N / 64:96:700/8128

Using a blank key list gave message FMS parameter error.

If you found a list but had no entries in it then using the list afterwards would give error message and end routine.

Load/Save 1 character names
Version/Machine: 1.07 / 64

One character names did not get .p appended.

Maintain other failure
Version/Machine: 1.07 / 64

Maintain other cannot evaluate string properly.

Command failing to evaluate properly, also clearing screen.

Maintain other new
Version/Machine: 1.09 / 64

Could not use the new command in maintain other.

Command required for dual drive systems.

Match dates using back arrow
Version/Machine: 1.0N / 64:96:700/B128

Could not match dates using backarrow to enter date later.

Match failure
Version/Machine: 1.0C / 64:96:700/B128

Match failed to find if fields contained / or &.

Match failure (sliding)
Version/Machine: 1.0N / 64:96:700/B128

Match found all records where numeric field = 0.

Sliding match said record matched if passed a numeric field of 0.

Menu waiting etc
Version/Machine: 1.07 / 64

When waiting for a key an invalid key highlights cursor.

New disk wrong drive
Version/Machine: 1.0S / 64:96:700/B128264

Did new on wrong drive.

New page failure
Version/Machine: 1.0B / 64:96:700/B128

@1,5"test"@3,5"test" did not force new page.

Should have given new page as cursor is past column 3 on line 5 after > printing first 'test'. Failure only occurs if second statement refers to the same line as the first.

No error message in format
Version/Machine: 1.0R / 64:96:700/B128

On some versions Error messages did not appear in format.

Messages: No key defined / No fields defined / Too many fields / Too many comments / Record too long.

No tone on 64
Version/Machine: 1.0Q / 64

On version 1.0P the tone would not sound.

Out of Memory
Version/Machine: 1.09 / 64

Run or load in a program did not clean stack.

Gave out of memory after many iterations as gosubs etc. left on stack.

Out of Memory error
Version/Machine: 1.07 / 64

Unexpected out of memory problem.

Caused by any conditional: nmat, pmat, eol, eof, if...then. All above required colons following them otherwise can give out of memory error.

Output all no parameter (date fields)
Version/Machine: 1.0Q / 64:96:700/B128

Output all with no parameters garbage after date field.

Got information from previous field if it was longer than the date field.

Output from list
Version/Machine: 1.03 / 64

If no fields specified then not using list.

First record came from list then all records from file afterwards.

Output gives field names
Version/Machine: 1.0A / 64:96:700/B128

Output to Device <> 8 if all fields gives field names.

Output numeric format wrong
Version/Machine: 1.0F / 64:96:700/B128

Numeric format of field incorrect when output.

Caused by format resetting due to plus command. If plus not used then ok.

Print command
Version/Machine: 2.02 / AII

Print command from menu failed if no parameters.

Print problems (4022)
Version/Machine: 1.00 / 64:96:700/B128

4022 did not like nulls being sent to it.

Printed lines backwards / no upper case etc.etc. Maybe affected other printers

Print/Display wrong position
Version/Machine: 1.07 / 64

Print affecting display position and vice-versa.

Interaction between screen and print positions.

Printer screen dump
Version/Machine: 1.07 / 64

CTRL p homing cursor after completion.

Printer wrong case
Version/Machine: 1.02 / 64

Printing to 1515/1525/1526 gives wrong case.

Cannot print lower case on 1515/1525/1526 printers as using sa255 added secondary address option to pdev command. Made default sa7.

Printing list cannot stop
Version/Machine: 1.07 / 64

Once printing a program could not stop it.

Printing numeric
Version/Machine: 1.06 / 64

Printing numbers across a page -- break not working.

If trying to print 9.62 and page break occurs at 9. then second page may show 6324567...

Prog editing crashes
Version/Machine: 1.02 / 64

System crashes while editing program.

If length of program at multiple of 256 characters then editing causes machine to crash. Cannot recover.

Prog first line number wrong
Version/Machine: 1.09 / 64

Unexpected error will not repeat if shift return etc.

Error happens after record stored.

Prog line > 128 characters
Version/Machine: 1.0E / 96:700/B128

Backarrow in Prog to go to previous line >128 then cursor left.

Failed to cursor left if line > 128 characters.

Protected prog will not run
Version/Machine: 1.0C / 64:96:700/B128

After protected program load and attempt list will not run.

Quit after error in format
Version/Machine: 1.00 / 64:96:700/B128

If error in format like record too long and stop pressed.

Left file in indeterminate state.

Record count high
Version/Machine: 1.07 / 64

Count of records from CAT in region 65530+.

Records contain trailing zeroes
Version/Machine: 2.02 / 64:96:700/B128/264

Trailing zeroes should not be stored on disk.

Rem or Data with square brackets
Version/Machine: 1.0L / 64:96:700/B128

Gave field name or brackets error.

Replica fields not set
Version/Machine: 1.0C / 64:96:700/B128

System did not copy to replica fields on some conditions.

Report generator
Version/Machine: 1.07 / 64

Report could generate command lines > 79 characters.

Could not edit long lines produced by report generator.

Report no file >8 character name
Version/Machine: 2.02 / 64:96:700/B128264:A11

If using Report from menus did not allow filename >8 characters.

Restore Failure
Version/Machine: 1.0E / 700/B128

Data restore not working on 700/6128.

Result failure
Version/Machine: 1.0K / 64:96:700/B128

[field40]+ something would not work.

You cannot define a calculation to be the 40th field + something else due to the internal compression of the field name.

RS232 on 700/B128 fails
Version/Machine: 1.0N / 700/B128

Trying to use rs232 printer on 700/8128 fails.

Error message i/o error £3 or system hang.

Screen dump with rmarg <>80
Version/Machine: 1.0M / 700/B128

Would not screen dump if margins > 80.

Select match
Version/Machine: 1.07 / 64

Select match not asking for parms after file.

Only occurs if match in progress then file swapped.

Select match where failure
Version/Machine: 1.03 / 64

Could not match if non-constant constant field.

Select match always looked for constant field to be default value. If constant edited then could not find it.

Set failure null string
Version/Machine: 1.07 / 64

Set failed if null string in file. Would fail if set file created by memo.

Setlink Invalid FMS Parameter
Version/Machine: 1.06 / 64

Setlink command will not work.

Setlink Invalid FMS Parameter
Version/Machine: 1.07 / 64

Setlink command will not work after database command.

Single drive 8250 backup
Version/Machine: 1.0Q / 64:96:700/B128

Single 8250 backup added.

Sort all using & failed
Version/Machine: 2.02 / AII

The & operator was not accepted in sort (Apple only).

Sort clears screen
Version/Machine: 1.0N / 64:96:700/8128

Executing a sort command cleared the screen.

Sort missing records
Version/Machine: 1.0S / 64:96:700/B128

Cannot sort if sort fields contain square brackets.

Fields should not contain square brackets. Not fixed.

Sort numeric failing
Version/Machine: 2.02 / 64:96:700/B128264:AII

Numeric sort fails on numbers usually integers.

Speed up loading files/progs
Version/Machine: 2.00 / 64:96:700/8128264

Rewrite of the sequential file handler.

Stop key not stopping print
Version/Machine: 1.05 / 64:

During top of page printing may be impossible to stop.

If plen/tien/title set to values such that title is always forced and print cannot continue, it is impossible to use stop key.

Subtotal not clearing
Version/Machine: 1.0E / 64:96:700/B128

Subtotal does not clear if comes after a colon.

Colon taken to be end of line even if in quotes.

Syntax error on backup
Version/Machine: 1.0Q / 64:

On version 1.0Q single drive backup gave syntax error.

Text fields only 251 characters
Version/Machine: 2.02 / AII

Would not allow text fields >251 characters.

Use of + crash
Version/Machine: 1.0C / 64:

Entering command using the + key if shifted crashes prog.

Shifted + key not a valid operator.

Notes

Notes

Superbase™ THE BOOK

Superbase is recognized as a leading database system for Apple and Commodore computers, with more than 100,000 users of 10 national language versions worldwide. Its menu-driven approach to file management makes it very easy to use, but the great appeal of Superbase lies in its built-in programming language, an extended version of BASIC. Mastery of this language is the key to developing sophisticated custom applications.

Superbase: The Book is the first in-depth guide to using the Superbase system, from first steps through to advanced programming techniques. Dr. Hunt shows the computer novice how to set up Superbase files, and how to search the database to extract useful and informative output. He describes how simple programs can help in automating a Superbase system, and explores many of the more sophisticated and powerful features with the help of programming examples. The wealth of hints, tips and examples makes **Superbase: The Book** essential reading for all Superbase users.

Dr. Bruce Hunt holds degrees from the University of Cambridge and the University of Calgary. A founder member of Precision Software, Dr. Hunt has been continuously involved with the Superbase project since 1983. This book reflects Dr. Hunt's wide experience of user's needs, refined in the Superbase seminars he has run in the UK, the USA and the Far East.

UK £11.95
USA $15.95

Commodore/Apple Computers
ISBN 1 85231 000 6