DATABASE MANAGER

(AtLast)

FOR THE AMSTRAD

CPC6128 PCW8256 PCW8512

OPERATING MANUAL

Authors ANDREW CLARKE & MIKE YORK

Advance Software Promotions Ltd
&
Rational Solutions

Contents

Chapter 1.

INTRODUCTION

The best way of learning about any program is to read enough of the manual to get an idea of what the program is supposed to do, and what it is capable of. Then it is important to try the program with the manual on your lap. First you will need to set up a working disc. Then try working through the examples in the text to feel your way round the program and gain confidence, referring back to the manual when you need help. If you ever start to get exasperated or irritated, then you may be trying to shortcircuit the learning process: Don't; it is worth taking the business of mastering a powerful program like AtLast slowly and carefully.

AtLast is a sophisticated program. It requires effort to master, but that effort is well worth while. Before you try to operate the program, you should have learned how to use your microcomputer. AtLast uses the CP/M operating system. If you do not know how to 'boot up' the CP/M operating system, copy a file or format a disc, then you should learn how to do so before going further. To do otherwise would be like trying to drive a lorry before knowing how to change gear. The microcomputer is not a particularly complicated machine, it just may be unfamiliar. There are no short cuts.

Although CP/M is not a very friendly operating system it would be silly to replace it by an alternative system within AtLast itself because this would be no help when you want to run another CP/M program and you would have to learn all over again. For better or worse, your computer runs CP/M. Get to know it!

Throughout the manual, the facilities described will be illustrated by examples which should be copied in order to get a feel for the methods by which databases are designed, defined and used. We will try to guide the user through the package, from the initial definition of a database, through adding new records and editing old ones, right up to the printing of forms and labels, using examples wherever we can. The first database is a name-and-address file for storing addresses and telephone numbers, and you are recommended to follow through its creation and use step by step.

Two other example database templates are included on your master disc. The first is for club membership the second is a simple cashbook accounting system. These examples serve to show off many of the features of AtLast.

You can copy the general way in which these databases were built up, in order to construct similar (and more useful!) databases of your own.

Like the program itself, the two main parts of the manual are broken down into two distinct chapters (5 and 6). The first (corresponding to the "DBDEF" command file) is devoted to the definition of a new database, the construction of different forms and the use of indexing keys, whilst the second (alongside the "DBUSE" command file) concentrates on using the database (i.e. adding records, editing old records, printing lists etc.) as well as the database repair facility.

Chapter 2.

GETTING UP AND RUNNING

AtLast runs on the CPC 6128 and PCW range of microcomputers. Versions are available to run on other CP/M microcomputers with a Z80 microprocessor.

AtLast needs a minimum program area of 52K. On unexpanded AMSTRAD computers it will therefore run only under CP/M Plus. A minimum disc capacity of about 170K is also required. The number of files, their record size, the number of indexes and the size of their entries that you can use are all affected by the memory available and the amount of data that you can have in a single database is limited by disc capacity.

AtLast needs a screen display of at least 24 rows by 80 columns.

Your master disc contains the following files:

The DEFINE database program	**DBDEF.COM**
	DBDEF.000
	DBDEF.001
The USE database program	**DBUSE.COM**
	DBUSE.000
	DBUSE.001
	DBUSE.002
	DBUSE.003
The CLUB example	**CLUB.DEF**
	CLUB.FRM
	CLUB.FIX

The CASHBOOK example	**CASHBOOK.DEF**
	CASHBOOK.FRM
	CASHBOOK.FIX
The keyboard setup	**ATLAST.KEY**
An example turnkey file	**ATLAST.SUB**

There may also be a file with a name like READ.ME or PRINT.ME. This may contain special information not in this manual. It is a text file. It can be viewed using the TYPE command or printed using a word-processor. It can also be listed using PIP:

```
A> PIP LST:=READ.ME
```

ATLAST.KEY applies only to Amstrad CP/M Plus computers and differs according to whether you are using a CPC or PCW computer. It is for use in conjunction with the SETKEYS command (see your computer manual) to configure the keyboard for AtLast.

ATLAST.SUB is relevant only to PCW computers and is an example file for using in conjunction with the SUBMIT command (see your computer manual) to automate the starting up of DBUSE.

Each of the two programs DBDEF AND DBUSE can be invoked by typing the name of the program you want at the CP/M prompt, followed by ⟨ENTER⟩ or ⟨RETURN⟩. E.g.

```
A> DBUSE
```

Advanced users should note that you may also follow this by an optional parameter identifying the database you wish to use.

All the relevant overlay files (DBDEF.00? or DBUSE.00?) must be on the currently logged disc for this to succeed.

To use AtLast you will not normally need all the supplied files on your working disc and you may need other files copied from your CP/M system disc.

Throughout this manual, certain keyboard commands are referred to such as ⟨Enter⟩ or ⟨Escape⟩. Generally, these will be invoked by pressing a single key or combination of keys on your keyboard. The correspondence between these commands and your keyboard depends on the computer you are using. Please see chapter 7 for the details of this.

AMSTRAD KEYBOARDS

On both CPC6128 and PCW8256, it is possible to customise the keyboard as you wish. The file ATLAST.KEY contains suggested settings and can be used by default. You will need to run the SETKEYS command every time you start an ATLAST session otherwise the keyboard commands recognised by AtLast will not function correctly. (In particular on a CPC, your ESC key will not produce the correct ⟨Escape⟩ code unless you have told it to.) To do this, enter the CP/M command:

A⟩ SETKEYS ATLAST.KEY

If you wish to change the keyboard keys that we have chosen for ATLAST, you can change the key definition file for SETKEYS called ATLAST.KEY to make best use of the keyboard commands. You will need to refer to the AMSTRAD manual to do this.

Please note that the ⟨Escape⟩ keyboard command used by AtLast is obtained on a PCW by the EXIT key. The special characters '\', and '|' are obtained by holding down EXTRA and pressing '½' and '.' respectively.

On a PCW computer you are also recommended to use the PAPER command

A⟩ PAPER n

if you are using continuous paper or labels according to where n is the length of your paper or labels in inches. This sets the gap (perforation skip) length to 0 and ensures that there will be no confusion between AtLast and the operating system over printer pagination.

AtLast uses a 24×80 screen on the CPC6128 and a 31×90 screen on the PCW8256.

To make up a working copy of the disc, you will need to copy all the files that you need onto a new disc. The simplest way to do this copying is to use the CP/M PIP command. If you are not yet familiar with this then you should read the relevant section of your computer manual or else the book "The Amstrad CP/M Plus" by Andrew Clarke & David Powys-Libbe. As there are several combinations of hardware, the files that you should put on your working disc for optimum efficiency will vary from system to system.

With either two disc drives, or a 'memory' (M:) drive, you can use the whole of a disc for your database(s). Your program files will lie on one disc and your database on the other. You should not keep your database on drive M: as you will lose the data when you turn off your computer. If the program is run from drive M:, however, it will work much faster.

PCW8512

On this computer, you are recommended to run DBUSE from drive M: and DBDEF (which you will use much less often, once you have a database in use) from drive A:. Your databases will be on drive B:.

The best layout for your working disc, therefore, is to put all the DBDEF and DBUSE files, the ATLAST.KEY and ATLAST.SUB file onto a disc with the files SUBMIT.COM, SETKEYS.COM, PIP.COM and PAPER.COM copied from your CP/M system disc. When you have booted up CP/M, put this disc in drive A:. The SUBMIT command enables turnkey operation in which the SETKEYS and PAPER commands are run automatically and the DBUSE program files are copied to the M: drive and executed. To run DBUSE enter the command

A⟩ SUBMIT ATLAST

To run DBDEF simply enter **A⟩ DBDEF** after having run SETKEYS and PAPER.

PCW8256

On this computer, your database will have to lie on a disc in drive A:. Your working disc should be set up as for a PCW8512 except that you should leave off the DBDEF files. These will have to share space with your database in drive A: until your database has developed to a state where you do not need DBDEF any more, or rarely, and then the DBDEF files can be deleted to leave space for your data. You must, of course, keep a back-up copy of DBDEF in case you need it again.

CPC6128

If you have a second drive, use this for your data. Your working disc, to be used in drive A:, should then contain all the DBDEF and DBUSE files, ATLAST.KEY and SETKEYS.COM from your CP/M Plus system disc.

If you have only one drive, then once you are happy with your database definition, you may transfer DBDEF to a separate disc, so that you have more space for data on the DBUSE disc in drive A:. If you like, these two logical discs can be opposite sides of the same disc. Should you need to re-use DBDEF, you can transfer the database definition files (with the file types DEF, FRM and FIX) between discs using PIP.

Chapter 3.

WHAT DOES ATLAST DO?

AtLast is used to store and retrieve information on a microcomputer. The sort of tasks that were traditionally done with a card box, such as membership lists, address lists, library catalogues, genealogy, client lists or academic references can be done much more simply with AtLast.

A computerised 'database' can be changed, copied or searched much more quickly than the traditional equivalent. Extracting information is much easier than is the case with a manual system. Even more importantly, AtLast can produce lists or 'forms' in a variety of different layouts for different purposes. If, for example, you had a simple name and address file, you could produce labels on your printer in separate batches according to postcode, you could print out invitations, or produce telephone lists in alphabetical order, omitting the address. It is this ability to produce print-outs of information in an infinite range of possible variety that makes databases like AtLast so generally useful, even if your needs are modest.

Despite the fact that it warrants a special computer jargon word, we meet databases in all forms in many situations. Really a database means any selection of information at all, from a jumbled heap of newspapers (which is really a poorly indexed database) to the human memory (which is a highly indexed database — for most people!). By computerising this information, we are doing the equivalent of writing out each piece on a card. What the computer does for us, is to find the right place, in the right file, in the right drawer, in the right filing cabinet of the right office. Then when the boss comes along at ten to five on Friday (as all mean bosses do) and asks for the appropriate piece of information, a few presses of a keyboard gets the data with slightly less swearing and frustration.

Sophisticated databases go a bit further than this, allowing the user to index a file in more than one way (e.g. indexing a book by author, by the title, by the ISBN number or even by the price) without having to keep four or five different copies of the data sorted in the appropriate way. We can also do more and change what the "card" actually looks like (title at the top, author at the bottom, the other way around etc.). AtLast allows all of these facilities.

What sort of information?

AtLast deals with 'structured' information. It cannot so easily store such unstructured material as extracts from newspapers, recipes, jottings, or lecture notes: Sherlock Holmes' famous scrapbook of crime would not have been amenable to AtLast. All the information in a particular 'database' must have the same structure. This structure is like the ubiquitous form that one is always having to fill in for bureaucracy. In effect, to use AtLast for a particular task, you have to design at least one form, and you will need to specify the type of information (whether, for example, it is a string of characters or a number) and its maximum length or size. Information is strictly pigeonholed and one has to decide beforehand on the size of the pigeonhole. If you were to do the equivalent task on paper, you would design a form on paper before you started, duplicate it, and fill in each form as you enter the information. You cannot use AtLast until you have decided how to arrange the information you are going to store.

The structure of a database is really the most important thing to consider about a database. Whether it is a manual card-index, or a 15 million record, cross-indexed database holding all of Britain's daily bank transactions, we must be clear about what information we want to include in the record, and what we don't. If, for example, we have an index of all the plants in a nursery, we might only include the name of the plant, and where to find it in the gardens: the database might be short, and takes up little room on either a computer disc or a card index, but it would do little to answer the awkward customer's question about where to find a 4' high plant that carries blue flowers in July, that thrives in acid sandy soil in shade.

In order to get the structure of the database correct, the question "what is it going to be used for?" has to be answered. If, in the situation of our plant database, you can be sure that the only use is to find the named plant when requested, then there is little point in including all the information about soil type, height, flower colour etc. But you must be absolutely sure that this information is not needed, because, as with most databases, changing the structure after some of the records have been entered, can be quite awkward, and even when that is done, each individual record has to be edited to include that new information.

The general rules are:-

a) Always be sure of what the information is going to be used for.
b) Make sure you include as much structure as you need (and don't be too worried about the record being too long unless you are in danger of running out of disc space — it's better than being too short).

c) If in doubt, include an extra field (see below) or two.

Let's take an example. Suppose we wanted to maintain a list of customers, with name, address, balance due, and the date at which we last updated the file. For a particular customer it might be:

Name:	**Mr R Supwards**
Address:	**The Manor**
	Mill Lane
	Stonham Aspel
	Suffolk
Balance due:	**234.89**
Last update:	**30/09/86**

This is one 'record'. Each customer will have its own record, analagous to a card in a card file. Each record (card) is divided up into 'fields'. A field is just a subdivision of a record. The balance due is a 'field' within the record. In this case, it is a number. The name is another field, this time a 'string' of alphabetic characters. The last update is another field and this time it is a date. The final 'field' in this 'record' is the address. This address is made up of four components or 'elements', each of which is a simple alphabetic 'string'.

You will notice that we have been careful to specify just what sort of information is expected to be put into each field. There are three good reasons for what must, at first, seem like an unnecessary complication. Firstly, one can get much more on a disc if you specify the data type; secondly, the program can perform special tasks on fields of a special type (e.g. totalling of numbers) and thirdly, you can check that the data being entered is reasonable. (If you enter the words 'lots' into the 'balance due' field, then that would be unreasonable!)

We have, you will have noticed, tried to make the layout of this record clearer by adding labels followed by colons, like 'name:'. We do not store these for each record because these labels will be the same for every record and therefore we need only store these once. With AtLast, we store these labels, and the layout of the information on the page or the computer screen, in separate files called 'forms'. For any database, we just store the data itself in the records, and we can arrange it in as many different ways as we like, in a collection of forms.

Before we get stuck into the task of entering the information, we need to make sure that we have left space for everything, we cannot scribble in the margin!

When deciding on your structure, you must also take into account exactly how you want it to be 'indexed'.

In a card file, the cards are usually kept in a particular order (e.g. alphabetical order of surname). If you wanted them in another order (e.g. balance due or last update) then you would have to take them out and sort them. This can be a slow and tedious process, even on a computer.

AtLast, rather than sorting the records after you enter them, keeps an index (or more than one index if you want) of your database. You will be familiar with the index in a book: The subject matter of a book is indexed in alphabetical order in the back of the book, and the number of the page where the subject is mentioned, introduced or explained is placed next to the name of the subject matter.

An index in AtLast allows the user to move straight to the part of the database where he or she wants to go. Unlike books, though, you can have any number of indexes. For example, we might index a group of plants by Latin name, common name or height. So when the database is defined, AtLast can be told to keep track of the order of the records by all of these criteria: By choosing one of these systems of ordering when using the database, the records are immediately in the appropriate order.

So far, it is all very uncomplicated. We have the basic idea of a very simple database. Once we have told AtLast what we want to do, we can put information into it and extract it in a variety of ways. This will be perfectly satisfactory for simple databases but most real applications are more complicated than this. Information tends to come in categories. Invertebrates and vertebrates; conifers, broadleaved trees, shrubs, hardy plants or alpines; principals in a solicitor's partnership; departments in a firm; AtLast is designed to manage categories with panache. If one can decide beforehand on the categories, one can store them much more compactly, and one is saved the bother of typing them every time one adds a new record.

One of AtLast's strengths is in the way that it handles this categorised data, so it is worth explaining this in more detail. Let's suppose that we have a collection ot Alpine plants, and want a database that will provide us with lists of propagation tasks every month, and will give us lists of plants of a particular flower colour, or plants that will flower in a particular month.

We are dealing with twelve months of the year, and there seems little point in having to record their name in every record, especially if we record the best month for propagation, the month in which the plant starts flowering, and the month in which it stops flowering. We could, of course, use a code (1-12) but it would be much better to let the program do the hard work by keying in the full names of all the months just once in the special-purpose first record of the database and letting the program convert what you type into a compact code, and converting it back when you wish to know what the code means. We also wish to record its habit (whether it is deciduous, evergreen, herbaceous, or bulbous-rooted). In most database programs we would key in a code and have to puzzle out what it means when we later access the database. With AtLast, we key in the alternatives once in a special record in the first file (the 'system record') and, subsequently, the program does the hard work for us. Doing this saves us a huge amount of file space, it saves a great deal of typing, eliminates problems due to typing errors, and makes the information much more readable. In this particular application, we also have a limited number of methods of propagation (heel cuttings, softwood cuttings, seed, root cuttings, or division of the root) so it is worth doing the same thing by selecting one of the methods of propagation that we have previously entered into the 'system' record. This sort of data, which is really a selection of one of a number of categories, is called a 'Constant' in AtLast since it refers to field data which is constant throughout a database however many times it is used.

Chapter 4.

NAD, A SIMPLE WORKED EXAMPLE

It is important to think through what one really requires in any one particular database. Once one has put a quantity of data into a database, it is not a trivial matter to alter the design of the database. It is worth making an effort to get it right first time. It is worth roughing out the design of a database on paper taking time and thought in the process, and one needs to be able to predict the uses to which the data will be put. Time spent at this stage will be saved over and over again when one comes to put the data into the database and manipulate the date.

Obviously a certain amount of confidence and experience of using AtLast is necessary in designing a database. This is best gained by following through the examples with manual in hand. For the rest of this chapter we shall illustrate the design and operation of a simple database.

After due thought, we have decided to have a simple system that replaces an existing manual address book. This consists of a name, followed by an address, with a telephone number. This uses few of the advanced features of AtLast but will introduce the method of designing a database.

We will definitely want to have an alphabetic list that we can occasionally print out to put on the wall next to the telephone, so we need to put surname (or company name) in a separate pigeonhole or 'field' to any first name. (Addresses are not often entered into real address books by first name and we want to have our list sorted in alphabetic order, surname first, like a telephone directory.) It would seem sufficient to have four lines for the address, as house names can be entered in the same line as street address if necessary. We need to put the postcode in a different 'field' as it is conceivable that we might wish to sort or search on the postcode and we save space in the database in view of the fact we need to allow less space for a postcode than a full address line.

It would be prudent to allow space for two telephone numbers (work and home). We have assumed that we will use the STD numbers rather than the name of the exchange, where possible, as we need to reserve less space for this (069 176 is better than Llanarmon Dyffryn-Ceiriog).

Having decided on the data we wish to keep, we also have to decide how we wish to access it. We will almost certainly want to call up any individual's record by name. We may also want to produce printed lists of telephone numbers in alphabetical order of names to keep on the wall by our telephone.

After chewing our pencil for a while, we can start to design our database. We switch on the computer, place the system disc in the drive 'A' and wait until the familiar **A⟩** appears.

Assuming that you have set up a convenient work disc or pair of work discs, as described in chapter 2, put your DBDEF work disc into drive A: and type **DBDEF** and then press your ⟨Enter⟩ key.

The following screen should appear:

```
        The Database Manager (AtLast)
   --------------------------------------------

           From RATIONAL SOLUTIONS

            (Level I, Version 1.22)

     Copyright (c) Mike York, 1985,1986

     Open what database ? [              ]
     ESC to Quit
```

If this screen does not appear, then it is most likely that you have not got DBDEF.COM and its associated overlay files (DBDEF.000 and DBDEF.001) on the current drive (try typing DIR DBDEF.* to see). Remedy matters and try again.

If all is well, you will see that you are being prompted to answer the question on-screen, **'Open what database?'**

Normally, when creating your own database, you will need to use your best creative endeavours to think of a name of a database. It must be eight or fewer letters. Shorter names will save time, so, in this instance, we will call our database 'NAD' (Name And Address). We assume that you have a two drive system and that you wish to keep the database on drive B:. We therefore type 'B:NAD', followed by depressing the ⟨ Enter ⟩ key. If you have only one drive, then simply omit the drive identifier 'B:' and type only 'NAD'. If you make a mistake press your ⟨ BackSpace ⟩ key to go back and delete the last character or move your cursor ⟨ Left ⟩ and ⟨ Right ⟩ keys to move to a particular position to overtype.

The program will now search for the file NAD.DEF on the disc. Since we have not yet created this database definition it obviously can't find it. The program is faced by a dilemma; Has the human being made a mistake, or has he/she really decided upon a new database. The program then tugs at your sleeve and asks: **'Define New Database (Y/N)?'**

Whenever you get this sort of question, you need only hit one key, either 'Y' for 'Yes' or 'N' for 'No'. (Computers do not allow 'up to a point' or. 'maybe'.)

Immediately you tap the 'Y' key, the disc whirrs, and a new screen appears.

```
   Database:  NAD

   FileName   Size  Can Edit  Can Delete  Can Add
 [SYS-NAD ]   0        T          F          F
```

This provides no information of significance, showing only that we have not yet defined a system file (nor need we in our simple example). We are interested in writing a file definition, so we move the cursor (shown by the brackets ' [' and '] ' and, if your screen display allows, highlighted) down by tapping the cursor ⟨ Down ⟩ key until it is on the line below the 'SYS-NAD' entry. Now we can type the name of the definition file which we will use. 'People' is what we shall call the file. It could be called anything, as long as it is eight letters or less. Type 'People' followed by your ⟨ Enter ⟩ key.

A new screen appears:

```
No FieldName Type      Elements Length Sub-length Sign Field[El]     Size
 1 [          ]
 2
 3

18
18
20
File:  People
```

This screen consists of rows and columns, and each row represents a 'field' entry. This is where we define all the information that is necessary for the file. Bear in mind that this is just an extension of what you would need to do with a manual form. Our first entry will be the name of the person. As we intend to produce alphabetic lists ordered or sorted like a telephone directory, first by surname then by forename(s), we need to separate the forename and surname (or company name). We therefore separate the 'Name' field into two 'Elements', forename and surname. It would seem an obvious choice to call this field the 'Name' so type the word 'Name' in the column entitled 'FieldName'. We tap the ⟨ Enter ⟩ key which moves the cursor onto the next column that requires filling-in. This is given the title 'Type'. This requires a little thought. All the data in our name and address file consists of words rather than numbers, and we have no categories. Our phone number is actually stored as a word rather than a number. We have therefore narrowed the choice to 'Alpha' or 'Upper'.

We would normally type names in mixed upper and lower case, so we type 'a' or 'A' for Alpha and press ⟨ Enter ⟩. The computer recognises that we mean Alpha, fills it in for us and moves to the next column entitled 'Elements'. We are being asked how many elements this field will have. We have already decided that the 'Name' field is in two parts corresponding to the forename and the surname. These parts are the 'Elements' of this field. (If you have ever programmed in BASIC, you will be familiar with Array variables. A field element is just like the element of an array in BASIC.)

We have two elements so we type '2' followed by the ⟨ Enter ⟩ key.

The cursor moves to the next column, entitled 'Length'. This means the length of each field element (or the length of the field if there is only one element). If we choose too small a length, then a particularly long name may be truncated, whereas if we choose too great a length then we will find that we will not be able to fit much data on the disc. Fifteen characters for a surname or forename seems a reasonable compromise so we type '15' and then press ⟨ Enter ⟩ . The next three columns, entitled 'Sub-Length', 'Sign' and 'Field [El] ' are all ignored as they are not relevant to Alpha fields. The computer fills in the last column itself, as it requires no further information on our part. This is the 'Size' column. If you filled the line in correctly, the number 32 will appear in this column. This is intended to give you an idea of the amount of space that a field will consume on disc.

We are now on the next line, poised to come to grips with the next field. This is going to be the address, so we might as well call it 'Address'. Go on, type it into the 'FieldName' column, followed by an ⟨ Enter ⟩ . If you make a mistake, the cursor keys can be used to move the cursor about in order to overwrite what you have typed. The backspace key rubs out the character that you last typed as it moves backwards over it. We will again choose 'Alpha' as the type. Addresses generally are up to four lines as we have already planned, so we will choose to have four elements. We type '4' in the 'Elements' column, followed by ⟨ Enter ⟩ . We are now in the 'Length' column so we need to decide what length each line of the address should be. A typical maximum line length for an address would be thirty characters. Type '30' and then press the ⟨ Enter ⟩ key. The computer responds by putting 124 into the 'Size' column and moving you to the next line.

We have, you will remember, already decided that the next field is the 'postcode' field. We give it the 'FieldName' of 'PostCode'. (Yes, you guessed it, we type in 'PostCode' into the 'FieldName' column.) Our postcode is not a number, but a string. Here, there is no choice of field type, as postcodes are always in upper case. Type 'U' followed by ⟨ Enter ⟩ . There can only be one element so enter '1' in the 'Elements' column. Postcodes are usually seven or eight characters but may be up to nine characters long, including the space separating the two parts, so we type '9' followed by ⟨ Enter ⟩ . We notice, from the final column that the computer has just obligingly filled for us, that we got away with using only 10 bytes.

The final line is now presented to use (we do not need to fill in all the lines!). This is, we already have determined, the telephone number line. We give it the 'FieldName' of 'Phone', the type is again 'Upper' (if we have to type in an exchange name we can be happy to do it in upper case). We allow two 'Elements' (home and work numbers). We choose to give each phone number a length of 14 characters, including the spaces. Your screen should now look like this:

```
No FieldName Type   Elements Length Sub-length Sign Field[El] Size
 1 Name      Alpha     2       15                               32
 2 Address   Alpha     4       30                              124
 3 PostCode  Upper     1        9                               10
 4 Phone     Upper     2       14                               30
 5 [       ]

20
File:  People
```

That now completes the definition of the fields. We can make any alterations or additions that may be necessary by moving the cursor around the screen using the cursor keys. When you have finished and the cursor is in the FieldName column, then we can press the ⟨ Escape ⟩ key (ESC on CPC6128, EXIT on PCW8256/8512) to go on to the next stage in our endeavours.

If all has progressed according to plan, the screen layout will have altered in order to show the index definition screen:

```
No FieldName Type    Elements | IndexName Duplicates Field[El]   Length Func
                              |
 1 Name      Alpha       2    | [         ]
 2 Address   Alpha       4    |
 3 PostCode  Upper       1    |
 4 Phone     Upper       2    |
 5                            |

20                            |
File:  People                 |
```

We must now decide how we are going to index our name-and-address database. This depends on how we wish to access individual records and in what order we wish to list them. In our case the most likely way we will want to do this is by name. We keep address books by name and we can always produce sub-lists using criteria such as the county, or town without choosing another index. The Post Office thoughtfully provided the postcode which often provides an excellent way of accessing an entry in the database, but we are not likely to index our entries using the postcode: It would only work for UK addresses and we may not know everybody's postcode. (In certain circumstances, sorting on the postcode can be useful as the Post Office make reduced charges for sorted letters. Telephone numbers also make surprisingly good codes to sort on in some circumstances as every one is unique to a particular telephone.) For our present purposes let's just organise things according to the name.

The first column to fill is the one headed 'IndexName'. Let's call it 'NADWho'. We type 'NADWho' and then press ⟨Enter⟩.

The cursor then moves to the next column that is entitled 'Duplicates'. This means 'do you wish to allow duplicates'. Duplicates are two or more identical key entries in the index. In the case of our simple address list we can allow duplicate names in the index. We therefore type 'T' (for 'True') in this column, followed by ⟨Enter⟩.

The entries which go into the index for any record are constructed from certain selected data fields and are called 'Keys'. We wish the first part of the key to be the surname. The field that we want to select from is the one we have already titled 'Name' so we type it in and press ⟨Enter⟩.

The cursor is now poised over the 'element' column (titled '[El]'). We now have to type in the element number, which for the surname is the second. We type 2 followed by an ⟨Enter⟩.

We want the surname selection to encompass the whole surname field, so we specify a length that is the same as the length of the surname element (fifteen characters). We therefore type 15 (and ⟨Enter⟩) in the 'Length' column and this moves us to the final column, with the name 'Func'.

The purpose of this is to specify any changes we wish to make to the field element in constructing the key. AtLast like most computer programs does all its sorting/ordering in ASCII order. ASCII (the American Standard Code for Information Interchange) is a system of numeric codes for displayed or typed characters. A table of ASCII codes is displayed in Appendix 2. The major point to note here is that in this system all upper case characters come before lower case. For example 'Z' precedes 'a'. Now this is not how we normally order names in telephone books.

As an example consider the problem of names like 'van Dyke': As his name begins with a lower-case 'v', he would be demoted to the bottom of the list, beyond the 'X', 'Y', and 'Z's. This is obviously not correct. Consider also names such as 'Mackeson' which should come before 'MacLaren', but in the world of ASCII, the upper-case 'L' in 'MacLaren' would cause it to be prompted to the top of the 'Mac's. We could have avoided this problem by defining the Name field to be of the type Upper rather than Alpha. But we have another way round this problem which enables us to keep the names in mixed case. This is essential if we want to generate mail-merged letters, for example our acquaintances would not be very impressed to receive letters like this:

Dear JOHN,

Give my regards to the whole SMITH family. . . .etc. . .

By applying a suitable function to the data field we can construct a key so that the index is ordered in a "case-insensitive" way ('a' and 'A' are treated as the same). This is done by applying the function 1 (convert to upper case). Since the conversion is only done to the key (the index entry) the data field is left in mixed upper and lower case. Note from the prompt on the bottom of the screen, when filling in this column, that two other functions are available. These will be described later. For the time being, we enter '1' into the Func column in the usual way.

We are next guided to the field column on the next line. This is done in order to enable us to select a second part to the key. Once we have got to the Smiths, for example, we would need to sort on the forename. We therefore specify the 'forename' element of the 'name' field as the second, and in our case final, part. This is done by entering 'name' again in the 'Field' column (although 'na' or even 'n' would have been sufficiently unambiguous for AtLast to know which field we meant) and selecting element '1' for forename(s). The computer suggests a field length of 15 characters for this part of the field which is just what we have available. We type ⟨ Enter ⟩ in order to agree with it. Again we want the key to be in upper case, just like the surname, so we insert a '1' into the 'Func' column.

That now wraps up the index and field definition phase for our file, so we press ⟨ Escape ⟩ (EXIT on the PCW). This takes us back to the Index Name column and another ⟨ Escape ⟩ takes us back to the list of files in the database. The new screen layout is the same as that which got us into the file editing process in the first place, but we have now got a file called 'People'.

```
Database:  NAD

FileName   Size   Can Edit   Can Delete   Can Add
SYS-NAD      0       T           F           F
People     198      [T]          T           T


T or F
```

We are poised in the column entitled 'Can Edit' and the computer offers the entry 'T'. This means 'True' (that we can edit entries). The following column also says 'T' or 'True' (that we can delete entries) and the same is true for adding entries. These are all as we want them so we press our ❬ Escape ❭ key.

We are then faced with a choice 'Save, Edit, Print, ESC'. We wish to save, so we press 'S'. The message 'Saved' should appear on the line above. Do not worry if the disc drive does not whirr at this stage. Under CP/M Plus a certain amount of disc buffering takes place. The relevant definition information will be written to the disc on or before leaving the DBDEF program. Now press ❬ Escape ❭. This saves the work we have done, and takes us to a new menu used for maintaining definitions:

```
AtLast : Database Definition Menu
-------------------------------------------------------------------------

  Database:  NAD

  1 ... Edit File Definitions

  2 ... Edit Form Definitions

  3 ... Autogenerate Forms

  ESC to Quit

  Enter Selection [ ]
WARNING: Do not remove data disc or switch off before Quitting
```

This is the full Database Definition Menu. When we created our new database this menu was by-passed because the program knew that the first task was to define the files (option 1).

We still have some work to do before we are airborne with this database. We have specified all the actual pockets or 'fields' of information, their sizes and their attributes, but we have not told the program how we want it all to appear onscreen, on a printout, or in a disc-based report file. In order to get started, we must create the layout onscreen. This will enable us to start keying in the data. Other forms we might need can wait till later.

The quickest way of doing this is to ask the program to do it for you, and modify the computers attempt to taste. This is what is meant by autogeneration of forms. Select this option by '3' followed by ❬ Enter ❭ at the prompt 'Enter Selection'. This does the Forms Autogeneration. The forms created are listed at the top left (in this case only one, called 'People'). Then the screen clears and in a moment, you are back to the Database Definition Menu. It is now time to clean up what the machine has attempted.

Type '2', followed by ❬ Enter ❭ in order to edit a form definition. The screen once more clears and you are asked whether you want to Get a form, Create a form, or ⟨ Escape ⟩ back to the previous Database Definition Menu. We want to modify the form that the program generated for us, so we type 'G'. The program then asks you for the name of the form to edit. You can have any number of forms, and so this is usually a sensible question. At this time it is not, as we only have one. This is the one called People. We can specify this lazily by typing a 'P', followed by ⟨ Enter ⟩. The program fills in the rest. If it got it right, then press ⟨ Enter ⟩ once more to confirm this and you will see a new screen layout:

```
Name : Name [1]           | Name [2]
Address : Address [1]                   | Address [2]
          Address [3]                   | Address [4]
PostCode : PostCode
Phone : Phone [1]         | Phone [2]

*************************************** End Of Section *************

--------------------------------------------------------------------------------
Form:  People      File:  People      Section:  Body

Edit, Head, Body, Tail, Save, Get, Delete, Rename, Create, ESC?
```

This is the form that was autogenerated for you. At the bottom of the screen, there is a list of options that you can select.

Edit, Head, Body, Tail, Save, Get, Delete, Rename, Create, ESC

You will notice that the computer has made an attempt at laying out your information in a form. It is not too bad, but could do with improvement. Each field is labelled by the name that we gave the field when we set up the file definitions. Every field that we specified in the 'file definition', has been allocated space in the form.
You may want to edit the computer's fumbling attempts at designing our form. You will probably want to include a bit more explanatory text in the fields. Let us forget the other options for the time being. Just type E (for Edit). The cursor will move to the top left of the screen. At this point, we have to release the firm grip that we have had on your progress, because it is now the time to alter this layout to your liking. This is done in just the same sort of method that you would use if you were altering the layout of a letter under a word-processor. The only difference is that intermingling amongst the test are "Field Markers". The keys to move the cursor round the screen, making alterations are designed to be the same as the ones used by common word-processors. We will mainly use the ⟨ UP ⟩, ⟨ Down ⟩, ⟨ Right ⟩ and ⟨ Left ⟩ arrow keys.

If you start to move the cursor round the screen, you will soon approach part of the screen containing a field marker where the actual information from the database is going to be substituted. At this point, the screen will change subtlely. The field has been highlighted and the prompt alerts you to the fact that you are on a field marker. It invites you to 'Edit, Insert text, Delete, ⟨Left⟩, ⟨Right⟩.' We are not interested in anything but moving left or right, so all that you need to do is to press your ⟨Left⟩ key to exit left or your ⟨Right⟩ key to exit right. This action gets you away, either to the left or the right of the 'field marker'. We can therefore move about quite freely round the program's attempt to generate a form and re-type bits we wish to change.

Typing characters or spaces will either cause them to overwrite the character already under the cursor or insert it before the character under the cursor. This depends on whether the "Insert Mode" indicator on the bottom line but one of the screen is 'Off' or 'On'. You can toggle between the two modes by pressing your ⟨InsertMode⟩ key. The character under the cursor can be deleted by pressing ⟨DeleteChar⟩ and the character to the left can be deleted by pressing ⟨BackSpace⟩. To insert an end-of-line character (to split a line) use the ⟨InsertLine⟩ key. When you have finished editing, ⟨Escape⟩ will take you back to the full forms menu:

Edit, Head, Body, Tail, Save, Get, Delete, Rename, Create, ESC?

There are other things that you can do to edit a form but we have only described the most useful ones which should be sufficient for a first attempt. See if you can alter the form to look like this:

```
Forename(s) : Name[1]          , Surname : Name[2]

Address : Address[1]
          Address[2]
          Address[3]
          Address[4]                                       PostCode : PostCode

Work phone : Phone[1]       Home phone : Phone[2]

........................................ End Of Section ...............
-----------------------------------------------------------------------------
Form:  People        File:  People        Section:  Body

Edit, Head, Body, Tail, Save, Get, Delete, Rename, Create, ESC?
```

If you don't yet wish to at this stage, don't worry. None of this is necessary. It's only purpose is to make the form look prettier. The autogenerated form was functionally adequate. (If you get into a mess, don't worry. Just use the option to Delete the form and then regenerate it using autogeneration again.)

If we have successfully edited the form then we will want to save it. We press 'S' (for Save) and the program will respond by asking us to confirm that we want to save the form. We reply with 'Y' (for Yes) and the list of alternatives reappears with the word 'Saved' appearing on the line above, confirming that the updated form has been properly saved to disc. (Again the actual write to disc may not have taken place yet because of the disc buffering.)

We can now press 'ESC' or 'EXIT' and the program is returned to the initial menu. We will do no more to define our database, so we can press ⟨ Escape ⟩ again. The program will write to disc and you will be returned to the opening screen asking you for a database name. Press ⟨Escape⟩ once more and you will see the CP/M prompt once more. We have now left the program.

We can now write some data into our address list database. Make sure that the disc containing the DBUSE files is in the current drive (usually A: or M:). We just enter the command

A⟩ DBUSE B:NAD

(or **A ⟩ DBUSE NAD** if we have only one drive)

and the following screen, displaying the Data-base Access Menu, should appear:

```
AtLast: Database Access Menu
--------------------------------------------

  Database:   A:NAD

  1 ... Add Records

  2 ... Scan/Edit Records

  3 ... List Records


  6 ... Repair Database

  ESC to Quit

  Enter Selection [  ]
```

If it does not, then it is likely that you are using the wrong disc or you misspelt something.

As you see from the menu, you have the choice of adding records (names, addresses and telephone numbers in our case), scanning or altering the contents of records, or listing records. Do not worry too much about the last option (Repair Database) as this is unlikey to be necessary. We are not at the stage that is equivalent to having a stack of cards with just the form printed on them. We need to type in some names and addresses so we press the '1' key, followed by ⟨ Enter ⟩.

After a bit of humming on the part of the disc, a blank 'People' form will be displayed and you will then be asked: 'Keyboard or File input?' You will be typing the address records from the keyboard so press 'K' (for keyboard) and you will see a new screen:

```
Forename(s) : [           ] , Surname :
Address :
                                                       PostCode :
Work phone :                , Home phone :

-----------------------------------------------------------------------------
Form:  People        File:  People        Records:     0
```

Notice that the cursor (delineated by '[' and ']' with the area between highlighted) is at the position of the first field marker on the form. Notice also that the name of the current form, the current file and the number of used records in the file (in this case, since we have not yet entered any data, this will be 0) are displayed below the form. You can now type in your names, addresses, and telephone numbers. The cursor automatically moves from one field to the next when you press ⟨Enter⟩. If you make a mistake, there are plenty of ways to correct them, using the cursor keys and other keyboard commands. When you have finished it, just press ⟨Enter⟩ until the following message appears at the bottom of the screen:

Save, Re-do, ⟨Up⟩, ESC

We want to save what we have done so we press 'S' (for Save). When the record has been saved (again this may only be to the internal buffer) the form will be cleared for the next 'card' with the cursor at the first field position again.

We go on entering names and addresses until we want to stop. Having saved the last record and with the cursor at the first marker for the new blank record, just press ⟨Escape⟩ to get the options:

Re-do, ⟨Up⟩, ESC

on the bottom line. Now press ⟨Escape⟩ again and we go back to the first menu.

If we want to leave the program we can type ⟨Escape⟩ at this point. Alternatively, you can try scanning for particular records that you have just typed in by selecting option '2'.

The program will display your blank form and will ask you for the index you wish to search on. Enter '1' for NADWho (which is also the default value you are offered) and the cursor will move to the first field marker on the form for a field element used in that key (in this case the forename(s) field element). Pressing ⟨ Escape ⟩ will bring up the first record in order of the index as well as an options menu. Note that the name of the current form, the current file, the number of used records in the current file, the current record and current key used for indexing are displayed below the form. The options given are:—

Form:	This enables you to change forms, to change the screen appearance, or even to change to another file within the database. With this database it is of no use as we only leave one form.
Key:	This changes to another indexing system. In this case the only alternative index is the record number.
Search:	This presents you with a blank record and allows you to enter all or part of the key of the record you are searching for.
Next:	Moves onto the next record in current index order.
Back:	moves to the previous record.
Print:	Prints out a copy of the current record (on the current form) to the printer, screen or file.
Edit:	Allows the editing of the current record.
Delete:	This allows the deletion of a record (you will be asked to verify if you really want to take this drastic action).

Pressing ⟨ Escape ⟩ (EXIT on the PCW) at the options menu, takes you back to the DBUSE main meny.

Option 3 at the main DBUSE menu allows printing of all, or selected, members of the database. The first thing asked for, is the index to use in ordering the records for listing. Having supplied the program displays the first and last keys (i.e. the key fields of the first and last records in the index) with the cursor positioned over them, so that you can determine where the list is to start and end. In effect, this is just like the search mode of the scan/edit option of the DBUSE section, in that you can input part of the key, to find the appropriate record.

For the time being we will assume that you wish to start at the first record and finish with the last. Just press ⟨ Enter ⟩ for every field displayed on the screen.

When prompted 'Normal or Reverse order?', press 'N' for Normal (we don't want Reverse) and 'A' for All (we don't wish to select). You must now choose whether to send the listing to the Screen (press 'S'), the Printer (press 'P') or a disc File (press 'F'). The file name will be asked for in the latter case, and the result will be an ASCII text file — which can be edited using a normal text editor . If you are listing to a printer, then you will be asked if you are using Single sheet paper (press 'S') or Continuous paper (press 'C'). The program then asks for page lengths and the number of lines to be printed on each page. This can be suited to fit labels or any size of paper. If both the parameters are set to 1, then no extra lines will be sent to fill up pages, so each line will be printed one after another. Unless listing to the screen, you are asked whether or not to send form feed characters (ASCII 12) at the end of each page. The last question you have to answer is whether you want a new page after each record. Unless you are printing labels or standard letters, this will normally be answered 'N' for no.

Having done this, the printing will begin. Pressing any character whilst printing will stop it and you will be asked if you wish to continue. (If the printer is buffered then this may not stop the printer immediately.) When the listing is completed pressing any key returns you to the main DBUSE menu.

Before leaving this database for good (by pressing ⟨ Escape ⟩ twice), try some of the other listing options, particularly the facility to select records. This is described in detail in chapter 6.

There are various ways that you can improve your NAD database and here are some suggested projects for you to try. If you are unsure about how to do them, take a look at the CLUB database definition on your master disc (making a copy onto a data disc first) and study the relevant parts of the next chapter.

1 Design a form that prints a series of labels on your printer. Make sure that there is only one space between forename and surname on printed labels. You will need to edit a field marker and set the option to 'Omit leading and trailing blanks' to True.

2 Design a torm that produces a file that can be read by BASIC or a word-processor for mail-merging. The normal format for mail-merge files is "comma delimited".

3 Design a form that produces listing of telephone numbers against names. Edit the 'Head' and 'Tail' sections of the form to construct headers and footers.

Chapter 5.

DEFINING A DATABASE

Having got to the CP/M prompt A⟩ , type DBDEF in order to run the definition section of "Atlast". You will then be prompted with the question:

Open what database? [ ]

where (as is the convention with the program) you will be expected to enter your answer between the square brackets. The combination of characters that you type in here must be a legitimate CP/M filename, optionally preceded by a drive identifier. In the case of a database to be called CASHBOOK on drive B:, simply enter 'B:CASHBOOK' followed by ⟨ Enter ⟩ . Note that once you have told AtLast which drive to look on for the database, you need specify it no longer; from now on it will look on the same drive for all the files that are associated with that database without being told. At this point, the program looks on the drive to see if the files are there. (Obviously, if you have specified a different drive to the current drive, containing the DBDEF program, then you must make sure the disc is in the drive before entering the name. If the disc is a new blank disc it must already have been formatted.) If the DEF file for your database is there, then it will be opened and the program will pass to the main Database Definition Menu. If it is not, presumably because you wish to define a new database, then the program will ask you if you want to define a new you have made a typing error. Type 'Y' to confirm that you wish to define a new database.

If you are defining a new database, the program does not take you to the main menu. In this case, there is no need, as the first thing that always needs to be done is to define the files and you therefore have no alternative.

If you are editing an existing database, then choose option 1 to edit the file definitions and option 2 to edit the form definitions. Please note that any changes you make to the file definitions may also imply that changes need to be made to the form definitions. It is also possible that your new definitions may be incompatable with any existing data. If you are making changes of this sort, but wish to preserve your existing data, then you will need to follow special procedures. See the section on how to re-organise a database in chapter 7 for details

Defining files

The database may be contained in more than one file (in fact, with "Atlast" it almost certainly has two or more files), so the next screen display met is one that allows us to name and edit each file of the database. In the case of an existing database, this can be obtained from option 1 of the main menu. A list of files in the database is displayed.

Even with a new database there is always one file already named. This is the system file, which is called SYS-XXXX, where XXXX are the first four characters of your database name. For the CASHBOOK database it is called SYS-CASH. This example database, whose definition files are on your master disc, will be used throughout this chapter for illustrative purposes. There may be up to ten files in addition to the special system file.

Whilst in this screen listing the files, you can choose to edit a particular file by moving the cursor (the square brackets) down to its name and pressing return at the appropriate place. By moving down to the first blank line, you can imput the name of a new file.

The system file has a special role which must be born in mind when defining files. It will only ever contain one record. This record holds all the Constant data that the database will use and the next values of any Serial number fields. This will be explained in more detail later. In the meantime it means that unless you wish to continually switch between the file you are defining and the system file, as you think of more Constant or Serial fields, you must have thought about the field structure in advance, so that you can define the system file immediately.

In order to edit or define the field structure of a file, press ❬ ENTER ❭ whilst the square brackets are around the file name in the FileName column. This will bring up a screen for editing the fields which, for the file Transact, will look like:

```
No FieldName Type      Elements Length Sub-Length Sign Field[El]   Size
 1 Dated     Date          1       8        2
 2 Number    Serial        1       5                   NextTran      2
 3 Account   Constant      1      15                   Account       1
 4 Heading   Constant      1      15                   Class         1
 5 Reference Upper         1      12                                13
 6 ChequeNo  Fixed         1       6        0       F                6
 7 Descript  Alpha         1      20                                21
 8 Amount    Fixed         1      10        2       T   6

20
File:   Transact
```

There are twenty lines for the fields. This means that you may define up to twenty fields in your file. However, since every field is actually an array of up to 99 field 'elements', the number of data items, in practice, can be much more than twenty.

To define or edit a field, you must locate the cursor on the appropriate line under the column headed 'FieldName', type in the field name if it is blank, or edit an existing field name if you wish and press Enter . The cursor will jump rightwards to the column headed 'Type'.

There are nine types of field that you can define (seven for the system file) and they are displayed at the bottom of the screen for you to choose from. The nine types are:

Alpha:	Any string of letters, numbers and punctuation marks.
Upper:	Any string of letters (which will automatically be converted to upper case when typed in)
Integer:	A number between −32768 and +32767
Fixed:	A number with a fixed decimal point and up to 11 digits.
Real:	Any real number up to 11 digits and in the range of ten to the power −38 to ten to the power +38. (Note this can suffer some rounding errors). Exponents (decimal) are written with an 'E' followed by a sign and the power of ten. Examples are .1234E−20 and −.5678901E+02.
Date:	A date in day/month/year format (DD/MM/YY) or YYYY.
HMS:	A numeric field to hold hours:minutes:seconds (or even degrees:minutes:seconds).
Constant:	An item selected from a list in a specified field array in the system record.

Serial: A numeric field whose value is taken from a specified field element in the system record of type Integer or Fixed which is subsequently incremented every time a new record is added.

For obvious reasons, the last two types are not allowed in the system record itself.

In the example Transact file, the field Number is of type Serial. This means that this field is never edited once the record has been saved. What is more, it's value is assigned automatically from the value in the field NextTran in the system record in the file SYS-CASH. Every time a Transact record is saved, the NextTran field in the system record is automatically allocated a transaction serial number in ascending order and which cannot be altered. The associated system field for a Serial field must have 0 decimal places (see below) if it is of Fixed type.

The field Account in Transact is an example of a Constant field. It may take on one of two values. These are the values in the elements of the Account field in the system record in the file SYS-CASH. These will be assigned when the database is used for the first time. Sensible values would be 'BANK ACCOUNT' and 'CASH IN HAND' to distinguish between transactions which pass through a bank account and those which are purely cash.

Another example of a Constant field is the field Interest in the file Member in the CLUB database (also on your master disc). This is used to specify which special interest groups, from a pre-defined list held in the Group field in the system record in the file SYS-CLUB, the member will belong to. In a computer club, for instance, these might have the values 'AMSTRAD', 'ACORN', 'SINCLAIR', 'CP/M', and so on.

The other field types are fairly obvious in their use. There may be some ambiguity over when to use Upper rather than Alpha. The deciding factors are (a) whether the person entering data may be unsure about whether to use upper or lower case, in which case the Upper type is less ambiguous, and (b) whether there is some over-riding reason for using mixed case as, for example, in the Name field of the People file in the NAD database defined in the last chapter.

The field type is selected by typing sufficient letters to identify it unambigously (the first letter will do) and pressing ❬ Enter ❭. The cursor then moves on to the next column to be entered. In all except Serial fields, this is the column headed 'Elements'. For a Serial field, there is only one element allowed. In this column you should enter the number of elements required for this field.

Each field may actually be a collection of a defined type. For example a person's address may actually be one field — but would have say four or five elements — one for each line. In the system file, each field used to store values used by a Constant field in another file will need as many elements as there are different choices for the 'constant' field in the database (for example 12 the months of the year).

Having specified the number of elements, the next field, except in the case of Constant, Serial and Date types is the 'Length' of the field.) The lengths of Constant and Serial fields are automatically assigned by their associated system field and the length of a Date field is automatically assigned according to its Sub-Length (see below).

The Length of a field is always its maximum character width on the screen or printer. The range of values it may be asigned is displayed at the bottom of the screen when the cursor is in this column. In the case of numeric fields it includes the space for sign, decimal point and exponent (four characters) if these are allowed. If disc capacity is not a problem for you, then be generous with these limits rather than sorry later.

The Sub-Length is used only for Fixed, Date and HMS fields. For Fixed fields it is the number of decimal places. For Date fields it is the number of digits allocated to the year. If 2 then the date will always be taken to mean this century. If 4 it will mean anytime in the period 0000 — 9999 AD. In the latter case, a date entered with only two digits for the year will still be assumed to be this century. Dates in the first century must be entered with preceding zeroes. For HMS type fields, the Sub-Length will be either 6 (for an hours:minutes:seconds) display or 3 (for an hours:minutes display.)

The 'Sign' column is only relevant to Integer, Fixed or Real fields. If true (enter 'T') then the numeric field can be either positive or negative. If false (enter 'F'), then the numeric field is automatically set up as positive.

The column headed 'Field[El]' is unused in the system file — but is used in other files to link Serial or Constant fields with their associated field in the system record. A list of the available system fields is displayed at the bottom of the screen when we enter this piece of information. Again we need type only sufficient characters for Atlast to identify which field we mean.

In placing the cursor under these column headings, Atlast is moderately intelligent and knows which information it needs for any particular data type. For example, it will not ask if an Alpha field should be signed. It will also carry out a certain amount of checking that the definitions make sense. However, there are circumstances where it may be confused, particularly if you change data types in mid-stream, so you should always check for yourself that the definitions you have made are consistent.

The final column, marked 'Size', shows how much space the entire field will take up in each record saved (in bytes).

Whilst editing the field attributes (that is anything apart from the field name) pressing ⟨ Escape ⟩ will move the cursor back to the FieldName column. Whilst in this column pressing the cursor keys will allow you to move up and down the screen to edit other fields. Pressing ⟨ InsertLine ⟩ or ⟨DeleteLine⟩ will result in inserting or deleting fields.

Pressing ⟨ Escape ⟩ whilst in the FieldName column will end field definition and move on to the index definition stage (except with the system file which has no index). An example screen, for the Transact file index definitions is:

```
No FieldName Type     Elements | IndexName Duplicates Field[El]   Length Func
 1 Dated     Date         1    |[TranDate]     T     Dated          5
 2 Number    Serial       1    |
 3 Account   Constant     1    |
 4 Heading   Constant     1    | TranNum       F     Number         5
 5 Referenc  Upper        1    |
 6 ChequeNo  Fixed        1    |
 7 Descript  Alpha        1    | TranRef       T     Referenc      12
 8 Amount    Fixed        1    |
 9                             |
10                             | TranCheq      T     ChequeNo       6

20                             |
File:  Transact                |
```

An index has two purposes. Firstly it provides a means to search, and obtain rapid access to, an individual record according to its indexed fields. Secondly it provides an ordering for scanning or listing records.

The simplest index is automatically present for every data file in an AtLast database. This is the record number, the position of the record in the sequential data file. (The record number can also be treated as though it is a field even though it is not declared or saved with the other fields in the file.) However, this is not usually practicable because, except in a very small file, one could never remember all the record numbers. We need to arrange something better. With AtLast, we can actually use up to five other indices other than the record number.

It is always best to have all your indexing systems worked out — in much the same way as having the structure worked out — long before you even approach the keyboard as, though more indexing systems can be added later, these take up some time.

An indexing system keeps the database in a predefined order, say alphabetical order or numerical order. With AtLast these indices are maintained up-to-date all the time, in contrast to many other database programs which require a file to be re-sorted whenever the index needs updating. In the CLUB database, the Member file is indexed on both the member's name and his/her membership number. In the CASHBOOK database, the Transact file is indexed by date, serial number, reference and cheque number. If you wished to list transactions in order of the Amount field, you could define an index to operate on that field.

An index is made up by entries containing selected data from a particular record, which may have been altered in some way which we will discover in a minute , alongside the number of that record. Searching the index, which is constructed in a way that is much more amenable to searching than the sequential data file, enables the extraction of the record number so that the record can be immediately and directly read from the data file.

Each index entry is constructed from the record data according to its 'key'. Defining an index boils down to specifying the key which is used to construct the index entry. In AtLast, each key may be composed of up to three field elements each of which may be processed in some way to improve efficiency. The resulting index is ordered first according to the first field element. If two entries have the same value here, then they are re-ordered according to the second field element. If there are still equal values, then they are re-ordered again according to the third element. In the example NAD database, we saw how the NADWho index was ordered first by surname then, for equal surnames, by forename(s).

On the index definition screen, the cursor starts off at the top of the 'IndexName' column. This is where we specify the name by which we shall refer to the index. (When the database is actually in use, you will be asked which indexing system to use. This is the name you reply with. It will also be the filename, followed by the extension IDX, of the file which will hold the index on the disc.) You can enter or edit the index name in the same way as a field name or file name.

The cursor then moves onto the 'Duplicates' column. Some indices may contain duplicate entries. Even after sorting on three levels you may have two indentical entries. There will be some circumstances where this is okay and others wnere it is not okay. If two people have the same name they would have the same index entry for an index composed purely from a name. If we wanted a unique record for every index entry we would have to specify a discriminating field in the key definition. If we don't mind duplicate entries then we simply enter 'T' for true in this column. On the other hand if we insist on every entry specifying a record uniquely then we enter 'F' for false. If we enter 'F' then AtLast will ensure that every record we add or edit is checked for a clashing entry in the index before allowing it to be saved.

The next column determines the first field element to be indexed on. Simply specify the field by sufficient characters for AtLast to recognise it and then, if it has more than one element, which element to use. If we forget which fields we defined we need only look at the left hand edge of the screen to see the complete list. Note that if we got the field name wrong, we can go back using the ⟨ Up ⟩ arrow key to try again — although we will have to enter a valid element number for the mistaken field first. In constructing an entry, Real type fields may not be used. Integer, Fixed, Date, HMS and Serial fields are all converted to numerical character strings which preserve their numeric or chronological order. Constant fields are converted to the element number corresponding to their value.

Next comes the 'Length' column. This indicated how many characters in this field element, starting at the left of the string , are to be used for the key. The maximum length for this part of the key is displayed at the bottom left. The entire index entry may be only thirty characters in length. For Integer, Fixed, Date and HMS fields, this is the number of significant digits in their character string representations.

The final column is the 'Func' column. This means the function to be applied to the key part before adding it to the full index entry. It is only used for alpha fields. The function takes one of three values:

0: As-is. This means there is case sensitive sorting, i.e. capital letters are considered to be before lower case letters. This uses the ASCII character set to determine which order the keyboard characters come in. There is a copy of the ASCII character set in appendix 2 of this manual.

1: Convert to upper. This converts the field to upper case letters before it is sorted (although it doesn't affect the case of the letters on-screen) This gives case insensitive sorting.

2: Abbreviate. This removes all the characters not in the range 'A' to 'Z'. This is useful for picking out initials.

These are listed at the bottom of the screen to remind you.

After the first key part (the first field element used) has been defined, the cursor moves to the next row in the 'Field[E I]' column and you can define the second and third key parts in like manner.

Pressing escape whilst in the 'Field[E I]' column brings the cursor back to the IndexName column where the cursor keys can be used to move up and down to define/edit more indexes. Up to five indexes can be kept on one file.

Pressing escape whilst in the 'IndexName' column will bring the program back to the file list screen. Note that the record size (in bytes) of the file we have defined is displayed in column 2. (This record size has a minimum of 8. Even if we define a record with only one field requiring only 2 bytes, the record size will still be 8.) The cursor is in the 'Can Edit' column.

AtLast provides three data protection methods. Any data file can be protected from having its records edited, deleted or added. This is specified in the three columns headed 'Can Edit', 'Can Delete' and 'Can Add'. If the DBDEF program is removed from the AtLast work disc then the user cannot change these settings. Each of these can be set to 'T' for true or 'F' for false. By default they are all set to true, except for the system file which cannot be deleted or added to.

Having specified these flags, the cursor moves on to the next line ready for the next file to be defined. AtLast allows up to ten files, beside the system file, in a single database. Although each file is logically independent, including several files in the same database means that it is easy to switch between them when the database is in use. This is particularly useful if different files contain related information.

Although AtLast does not have any special facilities for linking related files, care in defining fields can be very useful in using AtLast's facilities to relate records to each other. For instance, if two files have the same field defined, this can be very useful in selected listings of all records of one type which refer to another and if the record refered to is indexed on that common field, then rapid switching from the referencing record is possible in the scan/edit mode. This will be explained more in the next chapter.

Pressing ⟨Escape⟩ in the file list screen signals that we have finished the file definitions. In the bottom left hand corner of the screen, a choice of options should now appear asking whether we wish to Save, Edit, Print or ESC. Assuming everything is alright the definition can be saved and we can quit this screen using ⟨Escape⟩ once more. (Note if you try to quit without saving the definition, AtLast will ask you if you are absolutely sure that you want to let all that hard work go to waste).

If the print option is chosen a list of the structure can be printed to the screen, printer or to a disc file. The options involved in printing are explained in detail in the 'How to' chapter.

When you leave the file definition phase, you are presented or re-presented with the main DBDEF menu, the one that would be encountered first when editing an old database structure (rather than defining a new one). This menu has four options, editing the file definition (that is editing the structure — as above), editing the form definitions, autogenerating forms and pressing ⟨Escape⟩ (as usual) to quit.

Having defined the database structure, we still cannot use it until we have at least one form for every file in the database. A form is a screen (or paper) layout which will be used when editing, adding or listing records.

Defining Forms

Forms really specify the way that the data is arranged. It is not practical to have a database that merely lists out the raw data. In real life, we want the data embedded in text or laid out in a particular way on the screen. Naturally, if you are using a database heavily, you will need to have a number of forms for various purposes, such as creating standard letters, producing checklists or generating labels. AtLast calls all these things 'forms', and any file can have a number of them.

A form is a mixture of text and 'field markers'. The field markers mark the places on the form where data will be entered or displayed and the surrounding text has the purpose of explaining what this data is.

Each form has three sections, called Head, Body and Tail. Only the Body section is used for adding, scanning and editing data. The Head and Tail sections are used only when listing records. The Head section is listed at the beginning of each page of the listing. The main use of the Head section is in providing column headings for tabulated listings. No field markers may appear in the Head section. The main use of the Tail section is for reporting totals of numeric fields or the number of records listed. Totals will be calculated for any field element for which there is a field marker in the Tail section. Good examples of this are the 'List' forms in the CLUB and CASHBOOK databases.

You can have as many forms as you like for any file, assuming you have the disc capacity for the resulting FRM file. The capacity of a form is about 1,000 characters or 140 field markers or a compromise between these.

The quickest way to obtain some forms is 'Autogeneration'. Selecting option 3 after having defined a new database is always a good idea, as this will set the computer the task of generating a form, relieving the user from the tedious task of manually setting up a form — just for data entry. Having done this the computer will set up one form for each of the files associated with the database in use. For instance, in the CASHBOOK system it would set up a form for SYS-CASH and a form for Transact. The names of these forms would be the same as the files they are created for.

If a form already exists with the same name as its file then it will be left untouched and no new form will be created for that file. If you need to re-autogenerate a form, then you must either rename it or delete it first (see below).

Only the Body section of the form is created during Autogeneration.

The forms created are not as pretty as is possible if you edit them, but they do have the essential features necessary for adding and maintaining data records.

After autogeneration the program returns to the main menu. At this point the database can be used for entering and editing the information. If, however, you wish to pretty the forms a bit or make them more comprehensible to someone who was not involved in defining the database, or if you wish to create extra forms, then you should select option 2 from the DBDEF main menu. Of course, you don't have to do this straight away. Forms can be edited and created at any time, no matter how advanced you are with data entry.

The CLUB database has several forms defined for different purposes. It is a good idea to study them in detail to get a good idea of what is possible.

Having selected option 2, to edit form definitions, the program first asks whether you wish to Get an existing form (press 'G') or Create a new form (press 'C'). Creating a new form will immediately put the program into edit mode. Getting an old form will cause the Body section of the form to be displayed and present you with the standard forms menu. The options on this menu are:

Edit:	The enables you to edit the section of the form currently displayed. This is described in more detail below.
Rename:	Change the name of the form. A form name may have up to twelve characters of mixed case.
Get:	Get an existing form to edit.
Create:	Create a new form (using either the current form as a starting point or starting from scratch).
Save:	Save the current form.
Delete:	Delete the current form. This makes sure that, when you next use the database, this form is no longer available. You will be asked to confirm that you wish to delete the form before it is deleted.
Head:	Select the Head section.
Body:	Select the Body section.
Tail:	Select the Tail section.
ESC:	Returns to the DBDEF main menu.

The option you want is chosen by pressing the first letter of the choice, except for ESC which is selected by pressing your ⟨ Escape ⟩ key. Nearly all of these options are straightforward and self-explanatory, only the edit mode needs explaining in more detail.

In edit mode the form can be changed, much like using a very simple text editor. The complete range of key commands is available and they are shown in chapter 7.

The reasons for wanting to edit a form can be varied. Some examples are: that you want to make the field headings more explanatory; that you want to provide extra help to someone who would enter the data (e.g. a list of possible values for a Constant field); that you want to put column headings into the Head section for a tabulated listing; that you want to insert total markers in the Tail section; that you want to include special directions for another program that might take its input from an output listing from AtLast (e.g. 'dot' commands for a word-processor).

The total form size is limited to 1024 characters (not including continuous spaces and blank lines, which are considerably compressed). A field marker takes up to 6 characters whatever the size of the field on the screen. The form will scroll up or down at the bottom or top of the screen.

Whilst editing the form, you can indicate where a field element is to be put on the page by the use of your ⟨SetMarker⟩ key. This will put the current default field marker in the position of the cursor. This will be undefined (shown by a '?'), if the cursor has not yet passed through a field marker. If a field marker has just been deleted, then this will be the current default. Otherwise it will be the next element on from the last field marker passed through. (Either the next element in the same field array or the first element of the next field array.) Whenever the cursor is in an area occupied by a field marker, it is highlighted and you are given the following options:

Field Marker: Edit, Delete, Insert text, ⟨Left⟩, ⟨Right⟩

at the bottom of the screen.

The field marker can be edited by E. This will allow you to specify the name of the field (and if necessary, the element number) that you want displayed at that point. Typing '#####' as a field name in this instance, will place the record number at that point on the form (you only need to type the first '#'). In this way, the record number can be treated as an additional field. You will then be asked if you wish to omit leading and trailing spaces when listing and printing. This can be set to true (enter 'T') to ensure no unwanted gaps occur between fields when they are printed. An example of where you might use this would be in a label form, to prevent large gaps between forenames and surname. This flag defaults to false for an autogenerated form.

In the Tail section of a form, the field markers become 'total markers'. When the file is listed, the Tail section is printed at the end and any field marked in a total marker will have been totalled over all records listed and the total will be displayed here. If the total marker is '#####',(normally indicating the record number), then the total number of records listed will be displayed. See how this is used in the 'List' and 'ListTrans' forms for the CLUB and CASHBOOK databases.

Whilst on a field marker, pressing I will insert one space before the marker, leaving the cursor on the space where further text can then be entered as desired. This is mainly used for inserting text before a field marker at the extreme left hand edge of a line.

Pressing D will delete the marker.

Pressing your ⟨Right⟩ or ⟨Left⟩ arrow key will exit the cursor to the right or left respectively.

While in the text part of the form, pressing the ⟨InsertLine⟩ key will result in the insertion of and end-of-line character, causing any text on the line to be split at the cursor position while ⟨DeleteLine⟩ will delete everything up to and including the end-of-line marker, thus joining the next line.

Pressing ⟨ReMargin⟩ will cause text to be redistributed, one space between each word, between the left and right margins up to the end of the paragraph or the next field marker, whichever comes first.

Word wrap will operate as in a normal word processor whether in insert mode or not.

A backslash '\' at the end of a line will make sure that the carriage return at that line is ignored when printing, so that the following line is tacked on the end. This will allow the use of the full platen width on wide printers. The same form can still be used for screen listings without losing the display at the right hand edge of the screen.

Chapter 6.

USING THE DATABASE

Place your work disc containing DBUSE in the drive. Check that the files are on disc by typing DIR DBUSE.*. On a two-drive system, put the data disc in the other drive.

There are two ways to start up DBUSE.

One way is to enter 'DBUSE' at the CP/M prompt:

A ⟩ DBUSE

If all is well, you will see the same signing-on screen as for DBDEF, asking you to name a database. Enter the name of the database you wish to open, preceded by the drive identifier if you have more than one drive. For example, if you are running DBUSE from drive M: then your data disc will be in drive A: on a PCW8256 or drive B: on a PCW8512.

The other way is to enter 'DBUSE' at the CP/M prompt followed by a space and the name of the database you wish to open, preceded as usual by the drive identifier if the data is on a different disc to the currently logged disc. For example:

A ⟩ DBUSE B:DBNAME

In both cases, the message

Opening DBNAME.DEF

will be displayed at the bottom left, where DBNAME is the database name you have specified. The disc drive will whirr as AtLast searches for the named database. If it is unable to find it, then it will say so on the screen and you will be asked to enter the database name again. In this case you must try again or else press ⟨ Escape ⟩ to quit the program.

If AtLast is successful in opening DBNAME.DEF and its associated FRM, FIX files, then you will be presented with the DBUSE main menu. if you are using a system record then this will also be opened and read in to memory.

If this is the first time you have used a new database then the system record will be cleared (filled with blanks/zeroes) as it is set up. The first thing you must do is to fill in the Constant values and starting values for Serial numbers in the system record before you can add any data to a file. This is done using the Edit facility in the scan/edit option and will be explained in more detail later.

The DBUSE main menu provides you with the three main alternatives, and one subsidiary one:

1/ Adding records.
You can make new entries to the file, or one after the other. You can type them from the keyboard of read them in from a file that you have previously prepared. You cannot add a record to the system file.

2/ Scanning and editing records.
You can scan and search through the database, from any starting record in a data file, making alterations if desired. You can change forms or even files while scanning. You can print individual records to either the printer or a disc file. If necessary, you can delete them. If you wish to fill in, or alter the system record, you can use this option.

3/ Listing records.
You can obtain all or some of the records of a database and send them to the screen, printer or a disc file. You can choose to list out only those records satisfying certain conditions (eg: all travelling salesmen who live in Berkshire, over 45 years old).

6/ Repair Database.
Under certain circumstances (eg: a sudden power cut whilst editing a record), the database may be damaged. In most circumstances, it is easy to repair matters. You can also use this facility in conjunction with DBDEF to build a new index to existing data.

In normal circumstances, you will need to use only the first three options.

When you have finished using the database, please remember to make a back-up copy if you have made any changes. On Amstrad CP/M Plus computers this can be done using DISCKIT. On other computers, use PIP to transfer all the files that have been created or changed since your last back-up. To prevent disasters, remember to write-protect the disc you are copying from before copying.

Let's now look at the options available to us in turn.

Adding records

Normally, you will be typing records from your keyboard. This process is rather similar to using a wordprocessor. Unless you have only one form, you will first be asked to select a form you wish to use in order to add records. This will be one of the forms that have been created using DBDEF (see previous chapter). The program searches the FIX file for your current database for all the form names and displays them in a list on screen. All that you need to do is to type enough characters to uniquely identify the name of the one you want.(If the name of the form that you wish to use is the only one in the list beginning with a 'W', then typing a 'W' followed by ⟨Enter⟩ will result in the whole name of the form being displayed on the screen for approval.) If it cannot find a form that begins with the characters you have entered then you will have to re-enter the name of the from you want. Once the form you want is found press ⟨Enter⟩ again to confirm it and the (blank) form will be picked off the disc and displayed on-screen.

If the form you have selected belongs to a file which is marked with its 'Can Add' flag false (this will always be the case with the system file, for instance) then you will be told that you cannot add records to this file. On pressing any key you will be returned to the main DBUSE menu.

Once you have the form on-screen, you will then be asked whether you want to take your input from a file or the keyboard. Usually, when entering information into your database, you will want to key it in from the keyboard, This is not always the case. The data may have come from another computer, from another database, or even from another AtLast database. Nowadays, one can even get useful data from a bulletin-board or central databank. All these cases, where the information is stored electronically as a 'Text file', you use the 'adding records' option to enter it.

1/ Keyboard entry.

> Keyboard entry of text is designed to be as simple as possible (if the form has been well-designed you may hire a 'temp' to type all the data in). Press 'K' in response to the prompt and a new prompt appears, consisting of an 'open/close square-brackets', at each field or field element in turn. You can merely type in the information and move to the next entry by pressing the ⟨Enter⟩ key.

The program actually does some checking to make sure that you have not made elementary errors before it accepts what you have entered. Numbers must be valid; Date or 'HMS' fields have to be entered with delimiters to make sure that they make sense. ('/', ':', or '–' for dates and ':' for 'HMS' fields). Dates are validated according to the Gregorian calender (including leap years). If two digits or less are entered for the year, it is assumed to be the 20th century, even if four digits are allowed. Dates in the 1st century must have extra leading zeroes. It will be obvious what has been understood by AtLast, since the date accepted is displayed.

Serial fields cannot be entered. Their value is automatically assigned by the program to the value currently found in their associated field in the system record. This will be incremented when the new record is saved.

Constant fields must be entered to match the possible values in the system record. With Alpha and Upper type constants you need only enter sufficient characters to specify the one you want. AtLast will fill in the first one (in element order) that matches what you have typed in. If you have typed in nothing, then AtLast will offer the first element. If AtLast responds with the one you want, press ⟨ Enter ⟩, otherwise try again. You will not be allowed to enter an invalid value here. Be careful about spaces which are indistinguishable from unused characters after the last character. If in doubt, use ⟨ DeleteToEnd ⟩ after the last character you have typed. If you are likely to forget the possible values you can arrange for them to be displayed on the form, by editing the form (this involves using DBDEF) or, if there are too many of them, have a print-out of the system record beside you.

You can move back up the screen to the previous field using the ⟨Up⟩ cursor key and you can correct or edit what you have already done using ⟨ Left ⟩, ⟨Right⟩, ⟨ BackSpace ⟩, ⟨ DeleteChar ⟩, ⟨ Begin ⟩, ⟨End ⟩, ⟨ Tab ⟩, ⟨ BackTab ⟩, ⟨ DeleteToEnd ⟩ and switching between insert and overtype modes using the ⟨ InsertMode ⟩ key.

If the form in use is longer than will fit on the screen, then it will scroll up and down as required.

When you press ⟨ Enter ⟩ at the last field, or press ⟨ Escape ⟩ (ESC on a CPC or EXIT on a PCW) at any place on the screen, then you are invited to Save the record that you have just filled-in (press 'S') and go on to the next, Re-do the record from the top (press 'R') or edit it from the last field (press ⟨ Up ⟩). Another ⟨ Escape ⟩ at this point will return you to the main menu.

After each save, you will be re-presented with a blank form to fill in the next record. (Because of the disc buffering, a save will not necessarily be accompanied by an immediate write to disc.) When you have finished adding records simply press ⟨ Escape ⟩ twice to exit back to the main DBUSE menu.

As each record is saved, all index files that refer to the current data file are automatically updated. There is no need to do anything special to build an index when you have finished adding data.

If a record contains an invalid duplicate key, then you will not be able to save it until it has been edited.

Records will normally be added to the end of the data file which will grow as a result. If there is insufficient room on your data disc then the program will terminate after displaying an error message. If previously entered records have been deleted then these 'free' records will be re-used before new records are added to the end of the file.

2/ Data entry from a file.

Entry of data from a disc file is a very simple process. Firstly you will be asked to enter the name of the ASCII file containing the data text. This text is taken one byte at a time as though it is typed from the keyboard. Just as you type an ⟨ Enter ⟩ between field entries when typing records from the keyboard, so the text as taken from the file must have carriage-return/linefeed sequences between each field element.

The program does not distinguish between what is keyed from the keyboard and what it takes from a file, so the text in the file must be valid data only with no extra explanatory material. No extra 'carriage-return/linefeed sequence' is necessary between each set of field elements (i.e. each record) as the program knows when every field on the form has been filled.

All this sounds complicated, but all it really means is that each field, or field element (item of data) within a record must be on a line by itself in the text file (even if blank) and in the correct order required by the field markers on the form. The program eats omnivorously all that is in the file whether you intended it to do so or not. If the program suspects that you have made a mistake, (if a validation fails, or there is an invalid duplicate key) then it puts a brake on the process.

You will then be given the option to continue or abandon the transfer of data into your database.

Just as in keyboard entry, all Serial fields are entered by the program. They should therefore not appear in your input file.

Particular care is necessary with Constant fields, particularly over blanks/spaces. If you have not filled out the values in the system record with spaces, then any spaces that follow a data item in the input file will cause a halt and an invalid message.

Scanning or Editing records

One of the important requirements for a database program is to be able to browse or 'leaf through' a database, changing the data if required. This option allows you to do this.

First of all, the program asks you which form you wish to use. The program searches on disc for all the forms that you have prepared and displays them on screen. All that you need to do is to type enough characters to show the program which one you want. (If, for example, you wish to use the form 'Member' from the CLUB database, then typing 'me' followed by Enter will result in the whole name of the form being displayed on the screen for approval.) Once the form is found, it will be pulled off the disc and displayed on-screen. Otherwise you will be asked again for a form.

With the scan option, only the top part of the form is displayed if it is too large for the screen. If you have a very large record, then you should be careful to use a form that displays all the information you need to examine. If you edit a record then you can access the whole form. If you print a record to the screen then this is a way to see the whole form.

Once you have the blank form on-screen, then you will be asked to choose the index you wish to use. Index number 0 is always the record number itself. You will leaf through the file in the order determined by the index you are using. If, for example, you are using the individual's name in an address list to locate a record, then you will 'leaf through' the records in alphabetic order based on the name. If you have more than one index, you can choose which one to use. If you leaf though the database using the record number, then you will leaf through the records using the order in which they were originally entered — except where records have been deleted and replaced.

Normally, you will wish to start your scanning from a particular point in the database and so you will be asked at what point you wish to start. The program does this by asking you to fill in the key elements for the starting record. If, for example, you have a name and address database, and wish to scan through entries where the surname starts with N, you must specify this now by typing an N in this dummy or 'Mask' key entry.

The program rummages through the data until it comes up against the first entry that begins with an 'N' or after. This record will be displayed.

When a record has been displayed, there are a number of things you can do with it. You can switch forms, switch indexes, search for another record, go on to the next record, go back to the last record, print the current record, edit (change or alter) the current record or delete the current record. We will discuss each option in more detail.

Form. The 'Switch forms' option allows you to change the form that you are using. This may be necessary if, for example, you wish to print a particular record in a different format as specified by a different form. Imagine that you have selected a name-and-address database and found the entry for Roger Supwards. You now wish to print out a label for an envelope. You need to select a form that prints out the address in a layout fit for an envelope (you do not put the telephone number on it for a start!), and then you need to select the print option (see below). Taking another example, you have an entry in a file that contains a membership code. With this option, you can switch to a form used by your membership file, select the entry with the index based on the membership code and see the membership details for that individual. You can, with this option, leap nimbly from file to file within the database pursuring a line of enquiry.

Key. With the 'Key' option, you can change the index on which you select the record. If, with the CLUB database, we decide to scan on the membership number rather than the name, we do this by selecting this option and changing the index from MemName to MemNum. If the current record cannot be located in the new index, due to some inconsistency in the database (this can happen; see section on repairing the database) then the form will be cleared and you will be prompted to enter a key for searching, as in the Search option (below), to find a new start location for scanning.

Search. This initiates a new scan through the database, using a key value that you will be asked to enter for the current index. This means that, having done one search, you can initiate another.

If, for example, you are doing a series of address updates on a name-and-address database, you can find each name in turn, make the alteration, and then search for the next one. The same rules apply as for the initial search. This search activity finds the record corresponding to the first entry in the index on or after the key value that you type in. Remember that you must have the relevant key fields for the current index in your current form, or else you will not be able to specify adequately the key entry you want.

If you wish to edit the entry for 'Roger Supwards' (as he might have changed his telephone number), then you can specify this exactly in the mask key fields by typing 'Roger' in the forename field element and 'Supwards' in the surname element. Alternatively, if you know that there is no one else whose surname begins with 'Sup', then simply type 'sup', in surname and leave the forename field blank. The computer fills in the rest.

If you simply want to search for the first record in the file then simply press **⟨** Escape **⟩** or **⟨** Enter **⟩**.

If you see a message **'Record inaccessible'** then this means you are at record number 0 with no data. The message 'Record free' means that the record has been deleted and is available for re-use. This should only occur when scanning by record number.

Next. This allows you to display the next record in the database. The next record is the one that is next in the current index order. In the name-and-address example, if the current key is based on the name, then the next record will be the next one in alphabetic order of the name. This distinction is useful if one is trying, for example, to find the address of someone called Smith, or was it Smythe, or Smyth, whose forename one would recognise if one saw it. In this instance, one would select the 'Name' index, type in 'Smith', and move forwards or backwards through the records until the correct entry was located.

Doing difficult searches by inspection are a fact of life in using a database, and a great deal of time can be saved by using the right index to select on and making a good entry in the key 'mask' to search on.

Back. This allows you to display the previous record in the database. The previous record is previous in the current index order. It is the opposite to the 'Next' option and is used in conjunction with it.

Print. Whereas the most obvious use of this function is to get hard copy of the currently selected record, it is also useful for any purpose that requires the extraction of small amounts of information from the database. If, for example, you wanted to send one or two letters out to selected customers whose details were kept in a database, but you needed to inspect the record before deciding on who should be sent the letter, then you could prepare a special form beforehand, being the whole or part of the letter. When you have found the customer who will be the beneficiary of the letter, then you select your 'letter' form with the 'Form' option, and then print the form.

Usually, it is printed to disc, from where it is 'knocked into shape' and printed using a wordprocessor. If the letter is reasonably short and uniform, it can be printed straight to the printer. Printing to disc would allow a record to be sent via the telephone system using a modem, or to be read into another database. (One can write a number of records to disc using a form that conforms to the rules for data transfer, and then join them together using the concatenation option of 'PIP.COM' before they are read in to a new database.

Printing to screen is less useful, but allows you to see what a complete form looks like if it is longer than the screen area.

Edit. Editing a record allows you to change or alter a record's contents. You can only alter those components, or fields, or a database that are within the current form. This is actually exactly the same as adding a record as already described. ATLAST has several ways of editing text. Characters can be inserted, overtyped or deleted, and whole lines of text can be deleted in one keystroke (see appendix 1 for the details.) Obviously, if one needs to update the information contained within a particular record, this is the way to do it.

This option must be used when you first use a new database to fill in the Constant values and initialise the Serial fields in the system record. Remember, when filling in Constant values, that the first element acts as a 'default' when selecting a constant value. In some cases, it might be wise to leave this blank, or give it a name like NONE or OTHER to allow for circumstances where the information is unknown. Remember also that Constant values are assigned by the element number. If you change the order around then the values that appear in the data records that use them will change.

Delete. Deleting a record can be a saddening experience if it is done accidentally. To prevent this, the program will ask you to confirm deletion before going ahead. A detailed record will be replaced on the screen by the next record (with the same or a following index entry).

Not all these options are available for every file. For instance, you cannot change the index key for files that have no index defined. Nor can you search for a record in the system file as there is only one. Nor can you edit or delete records from files that are marked to prevent this. This 'marking' is made by DBDEF and consists of the flags 'Can Edit' and 'Can Delete' which can be set true or false.

Suppose you have a database set up as a catalogue of books. If you are the librarian you will want to be able to delete or edit records. But the last thing you want is for a reader to do so. Therefore you give the reader a copy of the database with a different definition file which is marked to prevent deletion.

Another example would be a transaction file for accounting purposes. Your auditor would be none too happy about the user editing or deleting transactions.

Note that the current record number is always displayed below the form (along with the current index key, form and file and the number of used records in the file). If the current record has a value greater than the number of records in the file, don't be alarmed. This normally means that there are some unused records in the file where data has been deleted. These records will be re-used when new data is added.

Once you have finished searching or scanning the database, the ❬ Escape ❭ key returns you to the main menu.

Listing Records

The ability to produce lists is the most important asset of a database. Lists can be sent straight to the printer, sent to the screen (console) or can be written into a disc file. At times, most database users will require to select lists based on a series of criteria. For example, the possessor of a database of plants may need to produce a list of all plants with red flowers that thrive in shade on an acid soil. At other times, a simple list of all, or part, of the records in their alphabetic order is all that is required.

The first task is to select a form. This is done in exactly the same way as for the previous two options.

If you have successfully selected your form, you will then be asked which index order you wish to list in.

The next task is to select a sub-range within that index. This can save a lot of time. If you are listing in name order and you only want to list people whose name begins with M then it would be very inefficient to start the search at A and go all the way to Z. The beginning and end values for the list are entered through the key field(s). By default, the begin value is the first in the index and the end value is the last.

If you are happy with these, simply press ❬ Enter ❭ once for each field element in the key mask. If, on the other hand, you wish to specify beginning and end values then simply fill in the mask fields in the same way as you search for a record in the scan/edit mode. The program will then present you with new default values for you to confirm or re-enter. In the case of the beginning value this will be the first entry on or after what you type in. With the end value it will be the last entry on or before what you type in. To select all names beginning with 'M', enter 'M' for the beginning value and 'N' for the end value (since you are unlikely to have a name just 'N').

Once the sub-range has been specified, you will be asked whether you wish to list the records in Normal index order (press 'N') or Reverse index order (press 'R'). (The first shall be last and the last shall be first!) This latter option can sometimes be useful if we wish to list dates in order of the most recent first, for instance. (For birth dates this would be youngest first.)

You are then asked if you wish to list All records in the sub-range (press 'A') or Select records according to further criteria (press 'S'). This last option brings you face to face with one of the most powerful features of ATLAST whereby it is possible to specify a list of conditions for any fields which must be satisfied if the record is to be printed.

Using the record selection feature, records are selected for listing by comparing them with whatever criteria you specify. For each condition you specify the field element you wish to select on, the comparison operator required and the value to be compared to. You can enter one condition per line for as many as you have room for on the screen.

When the cursor is in the column headed 'Field', at the bottom of the screen is a list of the names of all the possible fields for the current file. You now type in sufficient characters for AtLast to recognise the field you want. If the field has more than one element, then you must also specify which element you wish to make the comparison to. If you enter '0' here, then this means any element and the condition will be applied to all elements in the field, only one of which need satisfy the condition.

You must then enter the comparison operator. The possible values are displayed at the bottom of the screen:

BF : Before, or higher, in the sequence (higher or before in alphabetic order if a string of characters, less if a number, beforehand in date or time, or lower in element order within a constant field).

AF : After, or lower in the sequence (lower or after in alphabetic order if a string of characters, greater if a number, afterwards in date or time, or higher in element order within a constant field).

EQ : Equal to, or matching the comparison value.

EB : Equal to, or before, in the sequence (the same as a string or before in alphabetic order if a string of characters, less than, or equal to, a specified value if a number, beforehand or the same in date or time, or lower or equal in element order within a constant field).

EA : Equal to or after, in the sequence (the same or lying later in alphabetic order if a string of characters, greater or equal if a number, afterwards or the same in either date or time, or the same or higher in element order within a constant field).

NE : Not equal to, or not matching the 'comparison value'.

CN : Contains the 'comparison string'. This is only relevent for alpha or upper fields. If, for example, one were to explore a surname field in a name and address list, using the CN operator with 'bl' in the 'comparison value', it would select all those names containing a 'bl', including not only 'Bloggs', 'Blackshaw' and 'Blythe' but also 'Ablett', or 'Hubble' (all contain a 'b' followed by an 'l' within the field).

NC : Does not contain the 'comparison string'. This is only relevant for alpha or upper fields and is less useful than the CN operator. If the specified field within the record contains the string entered in the 'comparison value', then the record is omitted from the listing.

Comparisons are case-insensitive (no distinction is made between capital letters and lower-case letters) for string fields but otherwise they are in 'ASCII' order.

Once the comparison operator has been chosen, the next task is to put in the comparison value. Obviously, it must have the same data type (eg Alpha, Upper, Integer, Date etc) as the field it will be compared with. Constants can be specified in the usual way (i.e. just typing enough characters to define the constant element uniquely). Dates and HMS fields must be entered with their proper delimiters.

One can do a number of comparisons on the fields, and they are entered into the list, one at a time. Once the list is complete, one types ⟨Escape⟩ and the process of specifying the categories on which to sort the list, and the way that one sorts on those categories, is complete. One is then asked

Ok (Y/N/ESC)?

If all the comparisons have been entered satisfactorily, you should respond 'Y'. If you want to alter what has been entered into the comparison list, then type 'N'. An ⟨Escape⟩ at this stage stops the process dead in its tracks and returns to the main menu.

When a record is taken from the database, it is only listed if it satisfies all the specified conditions (they must all be true for the particular record).

Here are some examples of how you might use selection conditions to produce selected listings:

In the CASHBOOK database, for instance, listing transactions using the List form and the condition:

Account EQ BANK ACCOUNT

would enable you to obtain the bank balance from the total field at the end, while

Heading EQ EXPENSES

would give the total for the expenses heading.

In the CLUB example,

PostCode EA SW
PostCode BF SX

would select all members with a postcode beginning SW.

Address[0] CN London

would select all members in London whatever line of the address contains the word London. To prevent records with addresses like London Road being selected, you could specify the additional condition:

Address[1] NC London.

The condition

Interest[0] EQ AMSTRAD

would select all members in the Amstrad interest group.

Once the selection conditions have been specified (or by-passed by the 'All' option) we now have to specify the destination for the listing (P for Printer, S for Screen, F for File). In the case of listing to a disc file, you will have to enter the name of the destination file. If a file with that name already exists, then you will be asked whether you wish to overwrite it. If not, you will have to enter the name again. All that remains is to specify Single or Continuous paper ('S' or 'C'), the way that the listing is paginated (the number of lines per page etc.), whether form feed characters (ASCII 12) are to be used for page throws and whether one wants more than a record on a page and one is able to begin.

There are special parts of forms that can come into play when producing listings if they are specified in the form that has been selected for the listing.

These are called the Head and the Tail of the form. At the beginning of the listing and at the top of each subsequent page, the Head is printed. It is useful for titles or column headings. At the end of the listing the Tail is printed. The Tail is special as it is possible to print special fields consisting of totals. As the Tail is listed at the end of the selected records (the Body) it is useful for simple accounting purposes. (See the CASHBOOK example.)

Repairing the Database

This option is not so awesome as it sounds. It is not often necessary but when it is it can be very effective.

You may never need to repair your database. However, there is nothing worse than the awful feeling one gets when one finds that the database has gone haywire and there is no backup (it happens!). This option will generally only become necessary if something goes wrong. One may have entered information into a database when there is insufficient room on the disc; The disc may be faulty; One may have suffered a power cut before alterations have been saved on disc; You may have switched the computer off without leaving the program properly (i.e. pressing ⟨ Escape ⟩ at the menu), or you might have taken the disk out before leaving the program. One may get odd things happening; any changes you made might not be recorded properly; Records begin to disappear; Unfamiliar error messages may appear. Whatever may have occured, it is possible to rebuild the index files from the original data as long as the original data file is intact and, in some circumstances, it is possible to recover data which would otherwise be lost.

Another indication that the database needs repair is when the error message **'Error in key file'** is displayed during scanning or listing records.

The 'Repair Database' option actually attempts to rebuild the index files which constitute an important part of the structure of a database. It also rebuilds a linked list of free (and therefore re-useable) records within the file. When a file is saved, not only is the data you spent long hours typing in (or long minutes having piped in from an ASCII file) saved with it, but the program saves special information about what and where the data should be. If you have accidentally prevented AtLast from doing its essential housekeeping, the repair option is just the thing you need for trying to rebuild these files.

You can also use this re-indexing facility on a perfectly healthy database so as to add a new index to it. If a way of searching or listing becomes necessary and it was not predicted when the database was originally designed, then one can add it using the file definition facility of DBDEF to create a new key definition, save the new structure, return to DBUSE to repair the database, and ask the repair screen to reindex on the new key.

When you select this option from the main menu (press the '6' key, followed by a ⟨ Enter ⟩), you will be confronted by a display of all the possible files and keys that might need repair. You can ask to repair any or all of these by moving the square brackets to the letter 'N' by the filename and changing it to a 'Y'. You can then select whether to try and recover lost data and whether to re-build each index. When you have selected what needs repair, the program tugs at your sleeve for confirmation that you really want to go ahead. If you respond with a 'Y' then the program fusses around on disc, checking each record, rebuilding the pointer chains for free records, and (re)building the key files, and, as a final 'coup de grace', marking the end of each file that has been repaired.

Note that trying to recover lost data may introduce some mangled records which will then have to be deleted.

The best way to handle a situation where you fear that data may be lost is to make another copy of the damaged database. Run the repair facility on one copy without the option to look for lost data. If this results in lost data, then try again with the other copy, this time with the option on.

After the process has been completed, things should work a lot better. Occasionally, data may be irretrievable and may need to be re-entered by hand. Never mind, the basic integrity of the database has been restored.

WARNING:

If you have any records with illegal duplicate keys (this could arise from having made changes to the index key definitions after having saved them) then only the first of these records (in record number order) will be entered into the rebuilt index. The subsequent illegal records will still exist in the data file and can still be accessed for editing (to make them legal again from another index or the record number.

This repair facility is not infallible but it can always be augmented by those who are competent with public domain utilities such as DU.COM. With small files, SID.COM may also be helpful.

Chapter 7.

HOW TO . . .

1/ How to edit or enter a line of text.

At several points whilst running AtLast, you will be required to enter text. E.g. when creating forms, entering data, or making selections.

Enter a string of characters just as though one were using a typewriter or a wordprocessor program. When the line of text is completed one presses the ⟨ Enter ⟩ key. It is when the ⟨ Enter ⟩ key has been pressed that the program knows that the line of text is finished.

If you type an invalid character then a beep will sound. Valid characters depend on the circumstances. When enterering names for files, fields, indices or forms, the valid characters are all alphabetic characters, upper or lower case, digits 0 to 9 and the hyphen'-'.

You have a number of key commands that aid the entry of text. They correspond to special keys on your keyboard. The correspondence depends on your computer and this correspondence is defined below.

If you were the perfect typist, there would be less need for editing keys, but for the less accurate typist, there are plenty of ways of altering what is typed in; the electronic equivalent to 'snopake'.

In addition to the standard editing keys are certain special keys. One such is ⟨ Escape ⟩ . This generally exits from the current stage of the program returns you to an earlier stage or advances you to the next stage. It is usually a good key to press if you want to abandon what you have just been doing.

Another special key is ⟨ Break ⟩ which causes abrupt termination of the program. It is provided mainly as a panic button when you can't find some other way to abandon what you are doing. It terminates them in an orderly way. All the necessary disc updating is done before you leave the program.

There are four keys that control cursor movement: ⟨ Left ⟩, ⟨ Right ⟩, ⟨ Up ⟩ and ⟨ Down ⟩ . These keys will generally do what one expects. The ⟨ Down ⟩ and ⟨ Up ⟩ keys will usually have the same effect as ⟨ Enter ⟩ in addition to whatever else they do.

With AtLast, you can either overwrite characters or insert characters at the current cursor position. The ⟨ InsertMode ⟩ key toggles between the two possibilities. By default AtLast starts in overwrite mode with every new edit.

The ⟨ BackSpace ⟩ key actually moves the cursor back one position and rubs out the character on the cursor. This is the so-called 'destructive backspace'.

The ⟨ DeleteChar ⟩ key deletes the character under the cursor. More drastic deletion can be done by ⟨ DeleteToEnd ⟩ . This deletes from the current cursor position to the end of the line. It is particularly useful for making sure there are no unwanted spaces at the end of a string when selecting constants or specifying section values and also for removing default values that you don't want.

The ⟨ Begin ⟩ key moves the cursor as far as is possible to the left hand side of a the line or character string being edited.

The ⟨ End ⟩ key moves the cursor as far as is possible to the right hand end of the character string being edited.

The ⟨ Tab ⟩ means 'go to the next tab stop setting'. Tab positions are assumed to be set every eight columns. There is also a ⟨ BackTab ⟩ key that moves you to the previous tab position.

The ⟨ InsertLine ⟩ key is used when editing a form or a definition table. On a form, it inserts a new line at the current position, splitting the current line into two, if the cursor is not at one end or the other of the line. In a definition table, it inserts a new empty line ready for a new definition.

The ⟨ DeleteLine ⟩ key will delete a line from a definition table. When editing a form it will delete from the current position up to and including the end-of-line marker. If used at the end of a line it will cause the next line to be joined.

The last two keys are used only when editing forms. ⟨ ReMargin ⟩ will cause text to be re-laid between the left and right margins from the current position up to the next field marker or the end of the paragraph, whichever is soonest. The ⟨ SetMarker ⟩ key will insert a field marker at the current cursor position.

When editing a form, a normal word-wrap, as in a word-processor, operates to keep whole words or field markers together and respect the margins. The margins are the left and right hand edges of your screen.

The assignment of control codes to the keystroke commands is designed to conform as closely as possible to the most popular CP/M word-processing programs. On the Amstrad computers it is possible to set up your keyboard so that certain keys return these codes without having to resort to the usual ALT or CTRL modes. This is done by using the SETKEYS command. This makes the process of editing text much more intuititive. Example assignments are made in the file called ATLAST.KEY on the distribution disc and are listed here.

The command codes are:

Command	Code	Key(s)	PCW	CPC
Break	3	↑ C	STOP	CTRL C
Escape	27	ESC or ↑ [	EXIT	ESC
Enter	13	↑ M, Return	RETURN	RETURN
Left	19	↑ S		
Right	4	↑ D	Arrow keys	
Up	5	↑ E		
Down	24	↑ X		
Tab	9	↑ I	TAB	TAB
Backtab	2	↑ B	ALT TAB	CTRL TAB
Begin	17 83	↑ Q S	ALT EOL	F7
End	17 68	↑ Q D	EOL	F8
BackSpace	8	↑ H, Backspace	←DEL	DEL
InsertMode	22	↑ V	F1	F9
DeleteChar	7	↑ G	DEL→	CLR
DeleteToENd	17 89	↑ Q Y	ALT DEL→	CTRL CLR
InsertLine	14	↑ N	LINE	F4
DeleteLine	25	↑ Y	ALT LINE	F5
ReMargin	6	↑ F	RELAY	F1
SetMarker	11 84	↑ K T	PASTE	F0

Note that the '↑' before a character means that you should hold down the control (CTRL) key (or the ALT key on some machines).

2/ How to respond to menu prompts

Menu prompts are of three basic kinds. The first kind is where you are asked to fill in information between square brackets. This is done in the way described in the previous section.

The second kind is where a list of options is displayed on the bottom row of your screen. In this case, pressing the key for the first letter of the option you require will select that option , except for ESC which is selected by pressing ⟨Escape⟩ .

The third kind is where you are asked to confirm something. In this case you press 'Y' for yes, 'N' for no and ⟨Escape⟩ for ESC.

3/ How to use your printer

Whenever you wish to use the printer for producing printouts you will be asked a series of questions. Firstly, you will be asked if you are using Single sheet stationary or Continuous paper. If the former, respond 'S' and if the latter 'C'.

You will also be asked for the page length (in lines) you are using. This defaults to 66. If you enter a different value this becomes the new default. If you are using A4 paper it would be 70 if your printer prints at the normal line spacing of 6 lines per inch. If you are using labels then it will probably be 6, 9 or 12, depending on the height of your labels.

Next you must enter the number of printed lines per page. This defaults to 58. Again, whatever you enter will become the new default. The difference between this number and the page length is the number of empty lines AtLast will leave at the bottom of every page.

You will then be asked if you wish to use 'Form Feed' codes. These are special codes (ASCII 12) which instruct the printer to feed the paper to the top of the next page or eject the current sheet if using single sheet stationery. It is normally best to respond 'Y' for yes when using single sheets and 'N' for no when using continuous stationery.

After printing, AtLast will cause a page throw. If you do not wish this to occur, print with a page length and number of printed lines both equal to 1.

If you are printing labels, these must normally be single width labels. If your labels are organised two or three across the webb, AtLast cannot use the second or third labels except to print duplicates of the first (assuming you have set your form up with duplicate or triplicate sets of field markers).

Finally, if you are listing records, you will be given the option to give each record a new page. Unless you are printing labels or standard letters, you would normally respond 'N' for no to this question.

4/ How to reorganise a database.

It often happens that, despite the best of forward planning, one outgrows the design of a database. Although one can define new key fields, one cannot put new data fields into a database like AtLast. In particular, if new fields are inserted then the data in the file will get out of alignment with the definition leading to nonsense data.

Except for special circumstances, where field changes are compatible in type and length, you are advised not to alter an existing database. It is much better to create a new database and transfer the data to it. Imagine that you have a personnel file that maintains essential information on employees in a firm that is expanding. The management decide to reorganise the firm into new departments and implement a new career structure. The database is fairly comprehensive but did not predict the rapid expansion of the firm. There is no field for a constant giving the department and there is no field for a 'C.V.'. The new file is defined using the DBDEF program.

Firstly, make a copy of the existing database definition. You should transfer this to a new disc to avoid possible clashes between the names of old and new files. Then one can edit a copy of the existing file definition to add the extra fields. Obviously, one should think again about what would become necessary if the firm continues to expand, and one has to work out the consequences in disc storage capacity of expanding the record structure. Obviously, it will become necessary to modify a copy of the form definitions.

It is usually best to add new fields to the end of a record definition rather than insert them. (This is because AtLast refers to fields internally by the their number rather than their name.) Otherwise you may have to re-edit the index definitions and all the existing field markers in your forms and not just the ones you have changed.

The principle by which we transfer data is to export (list) the data from the old field to a temporary ASCII file on a disc and then import this data into the new file via the facility to add records from a file.

You must create a form in the old database that consists of all the fields that you wish to transfer, with one field (or field element) on each line, with no Head or Tail. They must be listed in the order that they will be required on the form which will be used to import the data into the new database. You must miss out any Serial field. Any fields which will be imported as Constant fields must be output with leading and trailing blanks omitted unless the new Constants have been set up padded with spaces. The form must contain only the field markers with no accompanying text or spaces or blank lines.

Then list the data to a file (making sure you have plenty of space on the disc) and use the option of page length 1, and lines per page 1 (so that no page breaks occur). You can then use the 'Add Records' option on the new database, reading in this disc-based report rather than taking input from the keyboard. In this way, all existing data is transferred to the new database via an intermediate report file.

Although this method may seem clumsy and tedious it is very powerful giving you the option to change field types if you wish (as long as the old and new types are compatible). For instance, you could transfer numeric to Alpha or Upper or vice-versa if the Alpha field contained only numbers. Or from Fixed to Real. Serial numbers could be converted to Integer or Fixed.

If the amount of information is larger than the capacity of the disc drives to cope, then it is possible to produce the ASCII disc-based listing using the sub-range or selection feature. If, for example, you had a large name-and-address file to transfer, then you could select just those with surnames beginning with the letters A-D, followed by E-H etc. There is no need to construct the new database in one fell swoop.

Since most databases have a facility to export data to an ASCII file you can use this method to transfer data even from another database program.

5/ How to calculate how much space a database needs.

A disc has limited capacity, particularly on the basic Amstrad PCW and CPC machines. You should become aware of the implications of your wonderful new database in terms of disc space. In other words, it is wise to know how many records you can squeeze onto the disc.

To find out how much disc capacity you have in all, make sure that all the essential files for use are present on disc, and there are no unnecessary files. Copy the SHOW.COM utility from the system disc into drive M: Then log onto the drive containing the disc on which your database is kept. Type SHOW (space) followed by ' ⟨ Enter ⟩ '; this tells you how many Kilobytes of disc space is left. If you multiply by 1024, then this will tell you how many bytes of space are left.

To find out how much space you need, enter the DBDEF program by typing

DBDEF followed by the name of the database followed by an '⟨Enter⟩'. go into the file definition function. The first screen lists the file definition. This gives the size in bytes of each record. One then has to make an allowance for the index files. Find the size of each key. Add these together and multiply by 1.5 to get a rough estimate of how much space is taken by the index file for each record. Like the data files the index files grow as records are added. Add the total space taken by the keys to the size of the record. Dividing the total number of bytes on the disc that we previously calculated using SHOW by this figure, we get an estimated figure for the maximum number of records that can be stored on disc. It is wise not to get too near this figure. If you are using more than one data file then the number of records of one type also depends on the number you will use of the others.

Disc capacity will also be a problem on single drive comuters if you wish to reorganise a database as you will need room for the transfer file (see previous section). It is essential to think about this in advance if you only have a single disc drive.

6/ How to relate records between files.

Most database programs are single file systems. That is only one record type is allowed and all the data conforms to that type. Other databases, including AtLast, are multi-file, in that several different record types are permitted in the same database, each type with its own data file and index files.

If you put common fields in different files then they provide a logical link between records in different files. For instance, consider a college in which each lecturer teaches a range of courses; suppose a maximum of five. We could set up a two-file system, one file for lecturers and one file for courses containing, at its simplest the following information.

Lecturer	**Course**
Name	**Title**
Lecturer number	**Course code**
Personal information	**Lecturer reference**
Course References (5)	**Description of course**
	Other details

Index on:	**Name**	**Title**
	Lecturer number	**Course code**

'Personal information' could be anything that was administratively useful, such as address, room number, phone number, pay details, and so on. 'Other details' could be hall number, time, day, duration and so on. The 'Course References' could be the titles of the courses or their references numbers. The 'Lecturer Reference', likewise, could be either the lecturer's name or their number. If there were few lecturers and all their names were unique, we could even set up their names as Constants. Likewise the course titles or codes.

(We could, of course, have set up a single file in which space was allocated in each Lecturer record for all the details of all five courses, but this would have meant a considerable wastage of space if a lecturer only taught one course. Alternatively we could have had a single file of courses, with all the personal details of the lecturer in each Course record, but this would have meant *duplication* of all the personal information in every course taught by the same lecturer. Suppose also we wished later to add a file of students with their personal details and references to the courses they took and possibly an extra reference to a lecturer who was their personal tutor. This would be next to impossible if we had not divided up the data to separate the lecturer data from the course data.)

Some databases, offer a means to automatically manipulate the references between files in various ways, such as providing automatic links between related records and merging records from different files in reports. AtLast is not quite as powerful as this, but, with a little thought it is possible to effectively simulate the best part of this within AtLast.

You will have noticed when using the scan/edit option from the main DBUSE menu, the facilities to (a) search for a given record and (b) switch forms and thereby switch files. This gives us a means to achieve relational referencing. Suppose we have a Course record on the screen and we wish to contact the lecturer involved. We can see the lecturer reference and we know that it is also an indexed key to the Lecturer file. Remembering the reference, we press 'F' to switch forms, select the lecturer form we need, select the index containing the reference field and type in the remembered reference. Up comes the lecturer record and we can read off his/her room or phone number to contact them.

Suppose now we wish to have a list of all the courses tought by a lecturer below the personal details of the lecturer. If we were content to list the course references, this could be done simply by printing the Lecturer record. If, however, we need more details of the courses to appear on the list, then we must find some way of extracting and merging the information from the Course records. This is where the powerful selection system comes into its own.

Firstly we use the scan edit option to find the Lecturer record. We print this out at the top of the paper, using the appropriate form, and specifying a page length just enough for this form (with, perhaps, a couple of lines to spare). The printer will rest on the next line on the same page. We now exit to the main menu and select the list option. We select the desired listing form, choose whichever index we wish to list in and the range and then 'S' to select. We enter only one selection condition, that the Lecturer Reference should EQual the one for the lecturer we have printed out. Then when the listing is performed the list of courses with the relevant details we need will appear below the Lecturer details. (Please note that this method will only work for a listing which will fit onto the first page. For subsequent pages, the page throws will be misaligned.)

Of course, relating records in this way is not limited to references between different files. It can also be used to good effect in relating records of the same type. In genealogy, for instance you may have only one file called People. The main index, of course, will be on their name. If you include a field each for mother and father, then these can provide relational references to other records in the People file which can be searched for by the index without even changing forms. To list all the children of a particular couple you select on a combination of Father and Mother fields.

7/ How to use the CASHBOOK example template.

If you have not yet looked at the CASHBOOK example template on your master disc, take a look now. The fields in the Transact file are fairly self-explanatory.

The Account field is intended to distinguish between cash and bank transactions. It is intended to have values such as 'CASH IN HAND' and 'BANK ACCOUNT'. If you want to expand the associated system field's element allocation to allow other values do so before you start entering data. You will also have to edit the Constants (system) form.

The Heading field is intended to allow for headings like INCOME and EXPENDITURE although these could also be broken down into sub-headings such as SUBSCRIPTIONS, POSTAGE and so on. Again you should decide how many you want of these and edit the system definition and Constants form appropriately.

Reference is used for recording invoice or receipt numbers or other references.

Amount is a signed field. Income will be entered as a positive amount and expenditure as a negative amount. If this convention is maintained, then any total obtained in a listing will represent the excess of income over expenditure.

Before you can enter any transactions you will need to fill in the system record with the Constant values and initialise the transaction serial number (NextTran).

The first transactions you enter must be the opening balances on the accounts.

In accordance with accounting practice, the file has been defined to prevent editing or deletion once a transaction has been saved. If you have entered a transactions incorrectly, you must make a 'Contra entry'. This will be indentical to the incorrect one except that the amount will be reversed in sign and the Descript field should be entered as 'Contra NNNNN' where NNNNN is the number of the incorrect transaction. You can then re-enter the original transaction as it should have been in the first place.

If you use the ListTrans form to generate listings, then selecting on the Account field will give the balance on that account and selecting on the Heading field will give the total for that heading.

At the end of an accounting period, and having taken all the reports you want, save your Transact file by using the CP/M command to REName it. You can then start afresh with an empty Transact file into which you will first enter the new opening balances.

8/ How to run the database from a different disc to the data.

See chapter two for the details on how the discs can be arranged. If disc space is short, it is wise to put all the program files on a different disc to the database. It is a good idea to put all the database program files on memory drive (Drive M:) if you have one. If the program files are in drive

A: and a copy of PIP.COM is on the disc, then typing:

```
PIP M:=A:DB*.*
```

will put the program files on drive M:. Because the M: drive is really fast memory, the whole program works must faster. On an unexpanded PCW8256 the M: drive is not big enough to hold both DBUSE and DBDEF, so you should copy across only DBUSE.* instead of DB*.*.

Do not use the M: drive for data. Unless you remember to copy all the files to a disc, any changes you have made will be lost when you switch off your computer or re-boot.

When using the program, log into drive M:, and then type DBUSE. When the program asks for the name of the database, you should precede the name of the database with the drive identifier and a colon. For example, a database called BOOKS would be B:BOOKS if the data is to be found on drive B:. Putting B: before the database name tells the program to look for all the files associated with the database on drive B:.

Use of the memory drive can be automated in a 'turnkey' system if you use the SUBMIT facility as described in chapter 2.

Appendix 1.
DATABASE FILE LAYOUT

A database consists of a group of files on a single disc. It is not possible to split a database across several discs without creating logically distinct databases. The database need not be on the same disc as the programs DBDEF and/or DBUSE.

The database definitions are contained in three files: DBNAME. DEF, DBNAME. FRM & DBNAME. FIX where DBNAME is the name of the database recognised by DBDEF and DBUSE. the DEF file contains the file definitions, the FRM file contains the form definitions and the FIX file is an index to the FRM file.

Every data file, including the system file has a name like FILENAME.DAT where FILENAME is the name of the file as assigned in DBDEF. Every index file has a name like INDXNAME.IDX where INDXNAME is the name of the index defined by DBDEF and employed in DBUSE to identify the index system being used.

WARNING:

If you have several files in the same database or several databases on the same disc, you must make sure that database, file and index names do not clash. The results could be disastrous if they do.

Appendix 2.

ASCII CODES

ASCII or the American Standard Code for Information Interchange is a code system for characters to be displayed or input on a computer. Nearly all microcomputers use this code. It only defines reproducible character codes in the range 32 to 126. Some computers use codes beyond this to reproduce other characters. AtLast will allow other character codes (in the range 160 to 255) to be used in form text or in Alpha or Upper fields. However, these may not be reproduced correctly on your printer if it is not set up to expect them. The ASCII codes are:

Code	Character	Code	Character	Code	Character
32	SPACE	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	£ or #	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Appendix 3.

ERROR MESSAGES

AtLast is a machine code program that has been translated from a high level language using a compiler. The compiler that was used does not allow full trapping of execution errors, except I/O errors. This could mean sudden termination of your program, possibly requiring you to repair the database (see chapter 6). However, considerable effort has gone into ensuring that such errors do not occur.

A possible error message you may get is the following:

Run-time error FO, PC=AAAA
  Program aborted

where AAAA is a hexadecimal address. This will occur if, for some reason, the program cannot find a program overlay file, possibly because you have removed the program disc, or the disc is faulty. Check your program disc before restarting.

Another possible error may occur due to as yet undiscovered programming errors in AtLast. Although the program has been well de-bugged, it is impossible to guarantee that this will not occur. If you get the following message:

Run-time error NN, PC=AAAA
  Program aborted

where NN is not FO, please note the values of NN and AAAA and report them to Rational Solutions, at the address on the cover of this manual with details of what you were doing.

There are other error messages which may occur but which are not so fatal. The message **'Duplicate key not allowed'** will occur if you try to save a record with same index entry as an existing record when duplicates are not allowed. Edit the record and try again.

If you get the message **'Error in key file'**, then your index has been damaged. Repair the database to rebuild the index in question.

The message **'Invalid. Re-enter'** may occur when entering data into a field. it is self-explanatory.

The message **'Insufficient memory'** is self-explanatory, you may have to reduce the number or size of the indexing keys you have defined for the file which was open when this happended. In extreme circumstances you may have to reduce the size of a file, or eliminate another unnecessary file from the database. This error will cause orderly termination, closing the database on the way.

The message **'DBDEF.OO? not found'** or **'DBUSE.OO? not found'** may occur when you first start up either DBDEF or DBUSE. It means that you do not have all the necessary overlay files on the program disc. Check the disc using DIR and try again.

There are several messages due to disc problems which will cause termination of the program in as orderly way as possible, attempting to close the database files to save the important pointers. You may have to take other action to save your data in extreme circumstances. These messages are:

Attempt to seek beyond end of file
Unexpected end of file
Insufficient room on disc to expand file
Insufficient room in directory
File full
Too many files open
File missing

sometimes followed by a file name and record number. These may not always make obvious sense, as there are additional complexities involved, however, they should give some idea what is going on.

INDEX