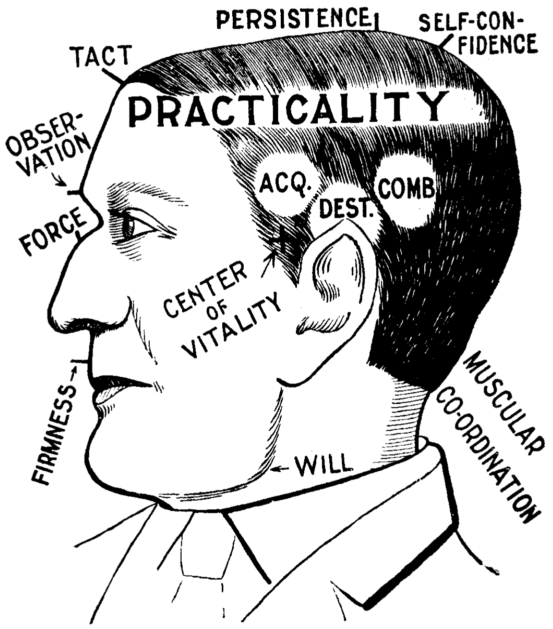
A.I.Q.



Ansible Information

A.I.Q.

Conscientious effort has been made to test this software in working conditions, but no guarantees can be made or claims for damages entertained. Genuinely faulty **A.I.Q.** disks will, however, be replaced free of charge on return of the original supplied disk.

If the software has been upgraded since this book went to press, more information on the DOS version may be contained in a file called READ.ME, which is in text format and may be edited through a word processor in the usual way, and perhaps printed out as a supplement to this book. More information on the Windows version may have been added to the Help screens — see **Release Notes**. We monitor our software after release, and we welcome your comments. If you encounter any serious problems, contact us at once. Please write to us at the following address:

Ansible Information
94 London Road
Reading
Berkshire
RG1 5AU
England

e-mail ansible@cix.co.uk
http://ai.ansible.co.uk/

This printing: *December 2008*
Ansible Information

Contents

1 • Introduction

Artificial Intelligence?

No. **A.I.Q.** does not attempt to create “a.i.” on your computer. What it does is to *randomize text*, doing so in a way that seems to produce meaningful results. By shuffling information stored in its “lexicons”, and presenting the results in a highly structured way, it generates an unlimited quantity of apparent wisdom and/or ideas.

Furthermore, it is completely “programmable” ... which is to say that you can develop your own lexicons from scratch, or adapt the ones we have supplied on this disk. Chapter 2 shows you how to use the lexicons we have supplied, and experimenting with these should give you an idea of the possibilities in the program. Later chapters explain how to create lexicons of your own, and explain in detail all the various simple (and progressively advanced) methods you can use.

No, you don't have to be a programmer! You need only be able to think up lists of words or phrases for **A.I.Q.** to permute. The lists can be very brief to begin with; if a lexicon looks promising, you will be able to add to it a bit at a time, as little (or as much) and as often as you choose.

What's the Point?

If you expect every piece of software to run your business, organize your life or write your books, then **A.I.Q.** is going to be a disappointment. It has very little “use”. (One or two of the lexicons provided—e.g., the short story seed generator—could be useful for writers, as a source of idea-germs and odd cross-connections rather than something that writes your novel for you.)

But it is an illustration of what a computer can be made to do with random information, and because it is completely programmable you will be able to write your own text files. As soon as you do, you will discover, as we did while developing the program, how intriguing, stimulating and (sometimes) downright funny the results can be.

Instal to your Hard Disk

Nowdays the easiest route, for both Ansible Information and our customers, is for us to email the software as an attached Zip file, AIQ.ZIP. Simply unzip the contained files into a folder of your choice and run the Windows version of **A.I.Q.**—program file AIQWIN.EXE. Otherwise, read on:

The **A.I.Q.** disk or CD-ROM comes with both DOS and Windows versions of the software. You should instal it as follows, depending on your operating system....

- ***Windows 95 onward:***

1. Insert our supplied floppy disk into Drive A, or our supplied CD-ROM into a CD drive.
2. If you have the CD and placed it in a drive which allows auto-running of programs, the Ansible Setup program should start automatically. If not, open the **My Computer** folder and then the appropriate drive: select and run the **Ansible Setup** program, whose icon is a flaming torch.
3. The Setup program allows you to change the directory where **A.I.Q.** will be stored, from the normal **C:\AIQ** to any other. You can also tell it not to create a new program folder containing the **A.I.Q.** icon. Change the settings if you wish to, and click the **OK** button.

4. The Setup program then copies the **A.I.Q.** files and will duly launch the program.
5. If you allowed Setup to create a new program folder at stage 3, you can henceforth run **A.I.Q.** by clicking **Start** and selecting in turn **Programs**, **Ansible AIQ** and **AIQwin**. If not, you will probably want to create a shortcut from some existing folder to (assuming you didn't also change the destination directory) C:\AIQ\AIQWIN.EXE.

- ***Windows 3.0, 3.1, 3.11, Windows for Workgroups:***

1. With Windows running, place our supplied disk in Drive A: and select **Run** from the **File** menu of Windows Program Manager.
2. Either type in the Setup program name **A:\SETUP.EXE** or click the **Browse** button and select first Drive A: and then **SETUP.EXE**.
3. The Ansible Setup program allows you to change the directory where **A.I.Q.** will be stored, from the normal **C:\AIQ** to any other. You can also tell it not to create a new Program Manager group containing the **A.I.Q.** icon. Change the settings if you wish to, and click the **OK** button.
4. The Setup program then copies the **A.I.Q.** files and will duly launch the program.
5. If you allowed Setup to create a new group at stage 3, you can henceforth run **A.I.Q.** by selecting this group and double-clicking the **AIQwin** icon. If not, you will probably want to create a a new program item in some existing group, to run (assuming you didn't also change the destination directory) C:\AIQ\AIQWIN.EXE.

- ***DOS 3.0 and higher:***

1. Copy the contents of the disk we have sent you to your hard disk. With the MS-DOS command prompt on the screen (normally **C:\>**), place our supplied disk in Drive A, and type:
   A: [Enter]
   INSTAL [Enter]
   The installation routine will create a directory on your hard disk called \AIQ, and then copy to it all the files you need for running **A.I.Q.** When this is complete, our disk may be removed from the drive and kept in a safe place.
2. Return to the **C:\>** prompt by typing:
   C: [Enter]
3. You can now run **A.I.Q.** by typing:
   AIQ [Enter]
   In future, simply go to the **C:\>** prompt and type the above line whenever you wish to run **A.I.Q.**

2 • Using A.I.Q.

A.I.Q. is run by the program file AIQWIN.EXE (*please note* that the full stops we use in the name of the overall package are not a part of the program filename). If you have followed the instructions in Chapter 1, this will now be accessible by double-clicking on the **A.I.Q.** icon in the usual way.

When **A.I.Q.** first loads you will see a blank screen with a standard menu bar at the top, and beneath this a toolbar of buttons -- most of them disabled, since before they become enabled an **A.I.Q.** *lexicon* must be loaded.

A.I.Q. lexicon files are the key to the program's use: in the form of lists of words and phrases, they provide the basic text which the program will randomize. The more you vary or add to the words and phrases in the lexicons, the more subtle and surprising will be the results the program produces. You are able to create your own lexicons from scratch, use the demonstration lexicons we have supplied, or customize our demos for yourself. Most of this manual therefore concentrates on working with lexicons, but this chapter provides a quick run-through of the basics of using the program. We begin with a note of the demonstration lexicons we have supplied.

THE SUPPLIED LEXICONS

A list of the lexicons can be displayed on the screen. From the menu bar at the top of the screen select **File** and **Open**; alternatively, click on the **Open** button on the toolbar. Either of these will produce a Windows dialog box, with a list of all lexicons found in the current directory. A lexicon file always has the extension **.AIQ**, to distinguish it from other files; thus the APHORISM lexicon (below) is contained in the file APHORISM.AIQ.

The supplied lexicons include the following (and you may discover others in the above list):

APHORISM	Put the calendar manufacturers out of business! Here is an endless supply of pithy observations, every one of them a reject from the **Oxford Dictionary of Quotations**.
BOOKLIST	Do you collect rare—*extremely rare*—books? This dustiest of book dealer's catalogues might be of help.
CAT	This is the simplest of all the lexicons on the disk. We use it in Chapter 3 as an example of a lexicon you can develop yourself. Running CAT at this stage will naturally not produce very exciting results....
DESRES	If you're house-hunting, you might be interested in this selection of house specifications, produced by the shiftiest estate agent in the world.
EXAMPLE	Demonstrates the use of various special **A.I.Q.** commands to generate individual random letters and numerals, force capitalization, and so on.
FANTASY	Ever noticed those fantasy paperbacks with wonderfully portentous titles? Ever wondered where the authors found their titles? Look no further.
FUSTIAN	This generates an endless stream of blank verse, “written” in the manner of the Immortal Bard himself.

GOSSIP	An unending stream of salacious tittle-tattle, all of it no doubt libellous.
HAIKU	The classic Japanese verse form, dating from the 16th century. Haiku generated by **A.I.Q.** are every bit as obscure as the real thing ...
HELP	You must have written a letter like this at least once in your life. Read a few of these, and perhaps spare a thought for the recipients.
HITECH	But here are some of the replies to HELP! You will certainly recognize the concerned tone, the clarity of the explanations ...
INFO	And here is the expert system from which the world's knowledge of computers arises. Everything you need to know is here, but finding it might be another problem.
LOTTERY	This simple lexicon generates 6 random numbers within the parameters of the British National Lottery. We ask for only 10% of your winnings.
MYTH	Throw away your encyclopaedia of mythology, legend and forgotten knowledge! Here are traditions so ancient the ancients had mislaid them.
PLOT	This is a “plot generator” for writers of short stories. It produces random combinations of characters, places, motives, conflicts, and so on.
QUATRAIN	Enigmatic verses which not only scan but rhyme. Move over, McGonagall.
RECIPE	Appalling recipes, to deter even the most committed foodie. When you have worked through this manual, take a close look at RECIPE and XXX, below; despite their deplorable output, they are the most complex and sophisticated lexicons in this package.
STARS	Instant horoscopes. One of the simpler lexicons, which could grow a lot.
STRUC	Want to write the kind of critical essay that no one on Earth can understand ... not even the most committed of structuralists? Let STRUC do it for you!
XXX	This spoof lexicon allows you to generate your own challenging “logic problems”. Solving them may however require a further program. As usual, you can modify copies of this lexicon to generate “problems” in various styles other than the slightly regrettable one found here. We keep wondering why our own efforts at new lexicons run so much to displays of bad taste.

The immediate “use” of **A.I.Q.** is simply to read the output from the lexicon files on the screen. (It is possible to automate the movement from one screen to the next; we describe how to do this later in this manual.) We hope you find the results amusing and interesting—and in the instance of PLOT perhaps even creatively useful—but the real interest of **A.I.Q.** is in creating your own lexicons. The rest of this book explains how to do this. Once you create a random-text lexicon of your own, you will discover the fascination of the program, and with practice you will be able to generate inspired and plausible random text.

A FIRST LOOK AT A.I.Q.

From the menu bar at the top of the main **A.I.Q.** screen select **File** and **Open**; alternatively, click on the **Open** button on the toolbar. Either of these will produce a Windows dialogue box with the list of all lexicons found in the current directory. Use the mouse pointer to select APHORISM.AIQ, then click on **OK**. The **A.I.Q.** icon or Windows 95 icon bar appears at the bottom of the screen, with the label APHORISM.AIQ.

Click on the button marked **A.I.Q.**, and after a short pause (while the program reads the lexicon) you will see a new screen. This will contain the text of an aphorism, together with details of who said it, when they said it, and sometimes the book or other source in which it was said. The important thing to note is that this aphorism has not been found in memory or in the file, but has been *randomly generated* from the huge list of possibilities in the lexicon. Regard it well, for you will never see it again!

At this introductory stage you need to know of only two optional commands from the random text screen:

If you select **More** from the menu bar, the program will generate another aphorism, completely at random. Every time you click on **More** the same will happen.

If you select **Cycle** and **Start** from the menu bar, you start the program automatically generating a new screen of random text, at intervals you can determine. The default time lapse is 5 seconds, but from the pulldown menu you can select intervals of 1, 2, 5, 10, 20, 30 or even 60 seconds. Halt the process with **Cycle** and **Stop** ... or **Exit**.

Selecting **Exit** from the menu bar will stop all random text operations, and return you to the main screen. The APHORISM.AIQ icon will be in place.

WHAT DOES THE LEXICON LOOK LIKE?

To examine the lexicon that is producing the aphorisms, you may use **A.I.Q.**'s built-in text editor. The lexicon contains the vocabulary that is being used (and so most of it consists of long lists of words and phrases), but in addition it contains formatting instructions, to give you control over how the randomized text will appear.

To look at the APHORISM.AIQ lexicon, simply double-click on the icon or expand the icon bar, and the lexicon text will appear. An alternative method is to click once on the icon, and then from the pulldown menu that appears select either **Restore** or **Maximize**.

The lexicon text fills the screen (the Toolbar button bar and menu bar remain in place). This is the **A.I.Q.** text editor.

At first sight the contents of the lexicon will look incomprehensible, a little like a programming language, with odd English words heavily interspersed with typographical and mathematical symbols. APHORISM.AIQ is one of our more ambitious lexicons, producing randomized text in several different ways. It remains in essence, though, a list of words and phrases, formatted to produce the aphorisms. In Chapter 3 we explain, with the simplest of examples, how the process of adding vocabulary to a lexicon takes place, and what those formatting commands actually do. If you follow that process through, you'll see how the numerous ways of randomizing text can be used.

For the moment, though, you can use the text editor to browse safely through the lexicon:

Use of the **Up** and **Down** arrow keys will scroll you a line at a time through the file. You can also use **Page Up** and **Page Down** to scroll through a screen at a time. **Ctrl-Home** will return to the beginning of the file; **Ctrl-End** will move to the end of the file. You can also use the Windows vertical and horizontal scroll bars to navigate around the file. All the usual Windows editing keys and clipboard operations are available.

While keeping the text on the screen, you can click on the **A.I.Q.** button to see the random text being generated, as above. Selecting **Exit** when you have finished will then return you to the text editor.

The **Save** button may be used to save the text of the lexicon, should you have made changes to it that you wish to keep.

The **Close** button returns you to the main **A.I.Q.** screen without saving.

The **Exit** button finishes your **A.I.Q.** session, and returns you to Windows.

OTHER WAYS OF EDITING A LEXICON

Each lexicon is in plain ASCII or “text” format. The most convenient way to edit a lexicon is through **A.I.Q.**'s built-in text editor, and this manual assumes that you will be doing so. However, the lexicons may also be edited using conventional text editors (such the MS-DOS program EDIT.COM or Windows Notepad), or any word processor in which it is possible to save documents in DOS text or ASCII format.

All lexicon files must have the extension **.AIQ**. For instance, STRUC is contained in **STRUC.AIQ**, BOOKLIST in **BOOKLIST.AIQ**, and so on. This extension is added automatically if you use our text editor, but you will have to remember to add it manually if working elsewhere.

The above information applies to the Windows version of A.I.Q. If you would prefer to use the older DOS version (AIQ.EXE rather than AIQWIN.EXE, a program which uses .DAT rather than .AIQ lexicon files), turn to Chapter 8.

3 ● How a Lexicon Works

One of the lexicons supplied on our disk is called CAT.AIQ. Start by opening it, as follows:

From the **A.I.Q.** main screen select **File** and **Open**; alternatively, click on the **Open** button on the toolbar. The Windows dialogue box with the list of all lexicons will appear. Use the mouse pointer or arrow keys to select CAT.AIQ, then click on **OK**. The AIQ icon appears at the bottom of the screen, with the label CAT.AIQ.

Click on the button marked **A.I.Q.**, and after a short pause (while the program reads the lexicon) you will see a new screen. It probably says:

```
The cat sat on the mat.
```

It might also say “The cat sat on the rug”, or “The dog sat on the mat” or “The dog sat on the rug”. If you click on **More** you will see these changes taking place. From time to time, the words “the” will be replaced with “a”.

To learn how to increase the “randomness” of a lexicon, we recommend you work through this chapter step by step. (If you try and take it all in at once, it will make your head spin.) We start with this simplest possible sentence, gradually increasing its complexity as we go along. This is the best way of learning the principles on which **A.I.Q.** is based.

THE FORMAT OF A LEXICON

When you have finished looking at the variations on “the cat sat on the mat”, move the mouse pointer to **Exit**, and click once. This will return you to the main **A.I.Q.** screen, with the CAT.AIQ icon showing at the bottom. Now double-click on this icon (or click once, and select **Restore** or **Maximize** from the pulldown menu that appears), and the complete text of CAT.AIQ will appear on the screen. What it looks like is shown overleaf.

The format of a lexicon must always be followed, and should be understood from the outset. Like all **A.I.Q.** lexicons, CAT.AIQ has three parts, in this order:

*Format instructions, which begin with a Star * command.* These are used to control the way the random text actually appears: it can be limited to one line, it can run on like text in a book, it can appear as a number of sentences in list form, and so on.

Sentence structures. This where you specify what you wish the random text to say, in skeleton format. You can have one sentence structure (as here), or several. You can make the program select any one of these at random, or you can make it work with them all. A typical sentence structure will have lots of words missing, replaced instead with instructions to the program to search the lexicon for words and phrases to drop in at random.

*Lists of words and phrases, again labelled with stars *.* These lists contain the words the program searches for, and selects at random. The longer the lists, the better the program will work.

All three of these parts will now be explained, using CAT.AIQ (over) as an example.

```
*header CAT.AIQ -- A.I.Q. DEMONSTRATION LEXICON
*limit 1

[The|A] =1 sat on [the|a] =2.

*Wordlist

*1 animals

cat
dog

*2 objects

mat
rug
```

Format Instructions — Star * Commands

The asterisk (or “star”) * is used as a “command” character, a **star command**. This is always placed at the extreme left-hand end of a line. Used this way, when a star is the first character in any line of text, it alters the effect of the rest of that line. In particular, when certain words immediately follow the star -- without an intervening space -- then that word (e.g. “header”, described immediately below) is the command. Asterisks can be used as normal characters elsewhere in a line. You should note the following):

- In the case of CAT.AIQ, the first line ***header** tells the program that all text following on that line (after the space that follows) will appear as a header at the top of the screen when **A.I.Q.** is running. (If you tried running CAT you will probably have seen this header in place.) The second line ***limit** tells the program that the text it displays should be restricted to one line. Users of older word processors will recognize the similarity of star commands to “embedded” format commands, not unlike Wordstar’s “dot” commands.
- There are several other star commands you can use as format instructions, and we list these in Chapter 5.
- If the star is immediately followed by a **space**, you can use the rest of that line to enter a comment or memorandum that the program *will not read*. E.g., you could insert a line like this as a memorandum:

  `* This lexicon produces variations on “The cat sat on the mat”.`

  This would have no effect on the program, and is never displayed when random text is being produced.

  Comment lines may be placed anywhere in a lexicon before the ***Wordlist** command (see below).

Sentence Structure

This provides **A.I.Q.** with the skeleton structure of the sentence(s) it is to use. In effect, any text placed in this part of the lexicon (provided it is not “blanked off” by being preceded by * and a space) will be read by **A.I.Q.** For example, if we placed here the sentence “All good men must come to the aid of the party”, **A.I.Q.** would wearily display the same basic sentence again and again.

Because we want to *randomize* the sentences, we can make the sentence structure include “variables”.

Here is the sentence structure as used in the lexicon file CAT.AIQ:

```
[The|A] =1 sat on [the|a] =2.
```

The significant characters here are the “square brackets” signs **[** and **]**, the “vertical bar” **|**, and the “equals” sign **=**.

The square brackets tell **A.I.Q.** that *text contained within them* is to be considered as variable. (Square brackets are standard keyboard characters.)

The vertical bar tells **A.I.Q.** *where the text becomes variable*.

For example, the sentence structure begins with **[The|A]**. This tells **A.I.Q.** that the sentence may begin with *either* “The” or “A”, and the program will therefore choose between them randomly. Both words are capitalized because they come at the beginning of the sentence. Later in the sentence structure, a similar variable **[the|a]** appears, with the same effect ... but this is not capitalized, because the words will appear in the middle of the sentence.

The vertical bar is found on most PC-compatible keyboards as the SHIFTed position of the \ or backslash key. It is often signified as a “tall hyphen” ... the bar with a gap in the middle.

A space or punctuation mark follows each word or variable word, as normal.

The equals sign tells **A.I.Q.** that the *character immediately following* is the “key” to one of the word lists. This is known as an **insertion command**. The first insertion command in this example is **=1**. This causes **A.I.Q.**, rather than report direct from the sentence structure, to search instead for a word in one of the word lists. In this case it is the word list labelled ***1**. Here it finds a choice between “cat” and “dog”. It chooses one at random, and displays it instead of the insertion command **=1**.

The same thing happens when it encounters the insertion command **=2** ... when it chooses randomly between “mat” and “rug”.

The words “sat on” are normal text, and appear in every sentence generated by **A.I.Q.** from this lexicon as it presently exists. The sentence structure ends with a full stop (displayed normally).

The point of this simple example is to show that everything in a sentence structure will be used by the program: every word, space and punctuation mark, as well as random variables. If it is ordinary text it will appear every time, no matter what. If it is made variable, either with two square brackets and a vertical bar, as in **[the|a]**, or if an insertion command is used, as in **=2**, the program will start making random selections.

Lists of Words and Phrases — *Wordlist

The third part of the lexicon is the lists of words and phrases, and this is headed by the ***Wordlist** command. Use of ***Wordlist** is fundamental to this program, and no lexicon must be without it.

The command must always appear on a line of its own: a star followed by the word “wordlist”. This can be in upper- or lower-case letters, or any combination of them: “wordlist”, “WordList”, “WORDLIST”, and so on.

After this command, any star followed by a character is the “key” to a list of words which the program will search randomly with insertion commands.

In the case of CAT.AIQ, two lists follow the ***Wordlist** command. The first is marked ***1 animals**, and the second ***2 objects**. (Only the character immediately following the star is important: the words “animals” and “objects” have been added as a memory aid; you'll find this useful when working with large numbers of lists![1]) These two lists are the ones searched by **A.I.Q.** when it finds the insertion commands **=1** or **=2**. The first contains the words “cat” and “dog”, the second “mat” and “rug”.

The format of CAT.AIQ should now be fairly apparent.

Any text between the opening star commands and ***Wordlist** is understood by the program to be a *sentence structure*. ***Wordlist** tells the program that the sentence structure has been completed, and that everything following consists of keyed lists of words which it may use for random searches.

In the next section we will progressively add to CAT.AIQ, making the text it produces increasingly random in nature.

DEVELOPING THE LEXICON — STEP BY STEP

Adding to the Word Lists

The greater the choice of alternative words, the more likely it is **A.I.Q.** will produce text that surprises you. Extra words may be added to the lists, almost without limit. (The effective limit is the memory available in the computer, but you are unlikely to run into this in most usual cases.)

First, you should have the complete text of CAT.AIQ (as shown in our illustration) on the screen. If you have left this, double-click on the CAT.AIQ icon from the **A.I.Q.** main screen, as described earlier.

Begin by adding extra animal names to the word list marked ***1 animals**.

Move the mouse pointer down to the line beneath “dog”, and click once. Now type in the name of any animal name that occurs to you: zebra, monkey, elephant, etc. Type it in lower case throughout, and be careful *not* to add an extra space at the end. Finish the word by pressing **Enter**.

Continue adding plausible animal names until you have twenty or thirty. Start each one on a new line, do not add an extra space at the end, and finish each one by pressing **Enter**. When you have typed enough, press **Enter** after the last animal name, so that a spare line remains between the last animal name and the heading for word list ***2 objects**.

(All blank lines, without exception, are ignored by **A.I.Q.** We suggest their use only to enhance clarity of layout, and ease of finding your way around.)

Now you can run **A.I.Q.** to see the effect this has had on the lexicon.

Move the mouse pointer to the button marked **A.I.Q.** and click once. (Or **Run A.I.Q.** from the **File** menu.) The text that appears will include one of the words you have typed in ... possibly including “dog” and “cat”, which of course remain available for selection. Click on **More**, and another randomized sentence appears. If the same animal names appear more than you expect, remember that the program is selecting *randomly* all the time, and just because a word has been used immediately before does not mean it won't appear again. Remember, the more variables you give the program to play with, the more it will find fresh text every time. (There are ways to increase or decrease the likelihood of words appearing, and we will come to that later.) For now, you might like to add a few more animals to the list.

Click on **Exit** to return to the text of the lexicon.

---

[1] Identifying the word lists as “*1 animals” rather than just “*1” makes it easier to use the lexicon editor’s short cut facility. A pop-up menu appears on clicking the right-hand mouse button: this menu’s option “= Codes” produces a list of all available word list (and other) commands, with identifying comments if these are present. See page 29.

Add a few more animals, and then add some words to the list of *objects* on which an animal might be sitting. These are contained in ***2 objects**. Move the mouse pointer to the line below “rug”, click once, and begin typing: table, chair, box, carpet, lawn, and so on. Again, stick to lower case, don't add any extra spaces, and always finish the word with **Enter**.

If you wish to see the effect this has had on the lexicon, click on the **A.I.Q.** button. Click on **More** to see variations, and click on **Exit** to return to the text of the lexicon.

(We won't repeat this simple instruction every time.)

Adding Extra Variables to the Lexicon

Let's add an adjective to the description of the animal, and an adverb to the verb. For this you need to create two extra word lists, respectively ***3** and ***4**.

With a blank line created after the last of the objects in word list ***2 objects** (press **Enter**), type:

```
*3 adjectives
```

(The word “adjectives”, or any other descriptive note, can be entered safely after the first character of a word list. Remember that the program reads only the character that immediately follows the star.)

Create a one-line space, then start entering adjectives that could plausibly describe any of the animals in word list ***1**: “large”, “hairy”, “wild”, etc.

When you have enough (that is, as many as you feel like typing for now), create another one-line space and enter:

```
*4 adverbs
```

After another blank line, enter a list of adverbs describing the *way* in which the animal might be sitting on an object: “lazily”, “placidly”, “threateningly”, etc.

When you have enough, finish the last word with **Return**, then move to the beginning of the document, either by using the **Up** arrow key, or by pressing **Ctrl Home**. The sentence structure can now be amended to take account of these new possibilities.

Changing the Sentence Structure

The new adjective (from the word list you have identified as ***3**) will go immediately before the animal identified by the words in list ***1** ... which is signified in the sentence structure by **=1**. In other words, amend the sentence structure by placing the insertion command **=3**, thus:

```
[The|A] =3 =1 sat on [the|a] =2.
```

(Note that naturally occurring spaces are left before and after each word or variable word in the structure.)

Now add the insertion command **=4** to the sentence structure, to place the adverb. This follows the verb, so should be situated between “sat” and “on”, thus:

```
[The|A] =3 =1 sat =4 on [the|a] =2.
```

Once again, you can have a look at how this has affected the result by clicking on the **A.I.Q.** button. Keep experimenting, by adding new adjectives and adverbs.

It is possible for words to be varied inside the sentence structure, without referring to a word list.

We have seen, for instance, that the sentence begins with either “The” or “A”, by enclosing these two words inside square brackets. Extra words can be added to these variables.

For example, we could add the word “Your” to the first set of square brackets, and the word “my” to the second ... thus introducing an occasional note of accusation to the sentence. To do this, simply add an extra vertical bar and the new word, *inside* the square brackets. Thus:

```
[The|A|Your] =3 =1 sat =4 on [the|a|my] =2.
```

You can add extra word-variables inside square brackets until most possibilities are exhausted. The program will happily work with several such alternative words in one set of brackets, but it's a cumbersome method when you want a large variety: instead, set up a new word list like those of animals, adjectives and adverbs already described, and use an insertion command.

Making Words Appear Less Often

Suppose that you wanted the adverb to appear only sometimes. As set out above, a typical sentence generated by **A.I.Q.** might be:

```
The hairy lion sat placidly on my table.
```

To make the adverb an unpredictable part of the sentence, you can reduce the frequency with which the program randomly “finds” it by enclosing it in one or more pairs of square brackets. One pair of square brackets, **[=4]**, will halve the likelihood of it appearing. So **[[=4]]** will give a one-in-four chance, **[[[=4]]]** gives a one-in-eight chance, **[[[[=4]]]]** gives a one-in-sixteen chance, and so on.

However, this introduces an uncertainty about the space that follows each word. To illustrate this, here is the *wrong* way of surrounding the adverb in square brackets:

```
[The|A|Your] =3 =1 sat [=4] on [the|a|my] =2.
```

This would reduce the likelihood of the adverb by half, but on the occasions when the program chose to leave it out, this would happen:

```
The hairy lion sat  on my table.
```

Note that *two* spaces now appear between “sat” and “on”. The correct way to make a word appear variably is to anticipate this, and include the space *inside the square brackets*. Thus:

```
[The|A|Your] =3 =1 sat [=4 ]on [the|a|my] =2.
```

When the program chooses to include the adverb, it does so with the space following, before the word “on”; when the program omits the adverb, the word “on” slides back so that it is next to the space following the word “sat”.

Any word, or any variable, or any set of variables, or any *part* of a set of variables, can have its likelihood reduced by surrounding it in square brackets.

It is also possible for variables *inside* variables to be loaded adversely, to reduce their likelihood ... or, in effect, to increase the likelihood of others.

To illustrate this, let's introduce an adverse loading to the last set of word variables: “the”, “a” or “my”. Suppose that you wished “my” to appear more frequently than either “the” or “a”, you would do this by *reducing* the likelihood of “the” or “a” by enclosing that part of the variable in one or more pairs of brackets. Thus:

```
[The|A|Your] =3 =1 sat [=4 ]on [[the|a]|my] =2.
```

This would give “my” twice as much chance of appearing as either “the” or “a”.

Pairs of square brackets may be added more or less ad infinitum, to change the likelihood of appearances. In theory, the upper limit is 255 pairs of brackets around any single word or variable ... but we suspect, in practice, most mere mortals will be content with just a few ...

Making the Word Lists Variable

In its present form our modest sentence is still confined to a rigid under-structure, irrespective of how many variables there are. That is to say, however much we tinker with it, the result will always come up with the thrilling news that some kind of animal sits in some fashion on some sort of object.

It's possible to reverse this (sometimes) by making the choice of word list *itself* a variable!

Let us suppose that you would like the sentence to say (occasionally) that one object sat on some other object. The insertion commands (characters preceded by the equals sign) can themselves be made variable. The name of the animal is at present selected by the insertion command **=1**; that of the object by **=2**. If we wished to give both of these an equal chance of appearing, we could enclose these in square brackets, separated by the vertical bar: **[=1|=2]**. The sentence structure would now be:

```
[The|A|Your] =3 [=1|=2] sat [=4 ]on [[the|a]|my] =2.
```

The same variable would make slightly more sense if there was twice as much chance of the animal's name appearing. In other words, if the logic was: [ANIMAL or ANIMAL or OBJECT]. In **A.I.Q.**'s syntax, this is achieved by placing a second variable against the first: **[=1|=1|=2]**. The sentence structure now becomes:

```
[The|A|Your] =3 [=1|=1|=2] sat [=4 ]on [[the|a]|my] =2.
```

This process of increasing the variability of the sentence structure can continue more or less indefinitely. For example, we still have not made any changes to the verb: we could vary "sat" with "lay", "squatted", "stood", etc ... we could also vary the preposition "on" with "beneath", "by", "next to", and so on.

("Next to"? Yes, two or more words can be included in a list, provided no extra spaces are placed *after* the phrase.)

Cross-referencing and "Recursion"

When creating a word list, you may include *references to other word lists*, including the present one! To illustrate this, here are two (very short) word lists labelled ***8** and ***9** respectively. The first contains soothing adverbs, the second contains disruptive adverbs:

```
*8 soothing adverbs
lazily
placidly
slowly

*9 disruptive adverbs
violently
angrily
threateningly
```

If word list ***8** is summoned from a sentence structure by the key **=8**, each soothing adverb has a one-in-three chance of being selected. Now let's disrupt the tranquillity a bit:

```
*8 soothing adverbs
lazily
placidly
slowly
=9

*9 disruptive adverbs
violently
angrily
threateningly
```

With the inclusion of **=9** in the word list, there's now a one-in-four chance that the selected adverb will be one of those from word list ***9**. To restore the tranquil balance a bit:

```
*8 soothing adverbs
lazily
placidly
slowly
=9
=8
```

```
*9 disruptive adverbs
violently
angrily
threateningly
```

The probabilities have changed again. One time in every five, the program's choice for **=8** will randomly come across the **=8** key, and select a reference to word list ***8**, sending it off in pursuit of a random selection in *the same list*. This might of course end up with a reference to word list ***9**, but the odds are against it. (It might even select **=8** again ... if so, it keeps hunting until it comes up with a real word.)

Another way of doing this would be:

```
*8 soothing adverbs
lazily
placidly
slowly
[=8|=9]
```

```
*9 disruptive adverbs
violently
angrily
threateningly
```

Now there is a one-in-four chance the program will find *either* a reference to list ***8** or ***9** ... with the same effect of sending it in pursuit of a real word.

Other sentence structure commands can be included in word lists. Words can be enclosed in pairs of square brackets to reduce the likelihood of being found. Two or more can be given variable status, by being enclosed in square brackets and separated by a vertical bar. Synonyms, defined by a star command (see page 23) can also be put into the word lists.

To introduce "recursion" or self-reference, you could enter the following:

```
*8 soothing adverbs
lazily
placidly
slowly
=8 and =8
```

This produces a one-in-four chance that instead of a single adverb, what gets plugged into the sentence will be "=8 and =8"—that is, a command to plug in two adverbs separated by "and". Hence you could get, at random, something like "The lion sat lazily and placidly on the mat." But of course there's a fair chance that plugging in the two adverbs will result in an instruction to plug in two more! Be wary with this technique, since it lacks conviction when a sentence reads "The cat sat slowly and lazily and placidly and slowly and lazily and slowly and slowly and slowly...."

There remain several further ways of varying the display of a simple lexicon like this. These are dealt with in the next two chapters.

4 ● Extra Sentences and Continuous Text

MULTIPLE SENTENCE STRUCTURES

You can create more than one sentence structure, and if you do the program will choose between them at random. (If you continue to use CAT, only one at a time will be displayed, so long as you leave the ***limit 1** star command at the top.)

For example, you could start by creating variations on **The quick brown fox jumped over the lazy dog**. You already have a word list containing animal names (***1**), so by adding more animals to the existing list you would increase the chance of randomness as the program selects words. The verb in this sentence indicates more activity than mere sitting, so you could create word list ***5** that contains further active verbs: “leapt”, “ran”, “climbed”, etc. A new list, ***6**, could contain more prepositions.

With this kind of preparation, the “quick brown fox” structure could look like this:

```
[The|A] =3 =3 =1 =5 =6 [the|a] =3 =1.
```

From this skeleton, you could start adding extra possibilities and variables.... Do you need *two* adjectives in front of the first animal name in the sentence? (One could be placed in pairs of brackets, to reduce its likelihood.) Would you like the second animal name to be, occasionally, an object, not another animal? (Introduce variable loading, as described earlier.) And so on. If you wished, you could go on and add several more sentence structures.

Although the rather terse “syntax” of a sentence structure looks very confusing at first glance, if you start it simply, and add to it progressively, pausing to run the program from time to time, you will find it a simple procedure.

CONTINUOUS TEXT

To make the text “continuous”—in other words, appearing as if in a book, with one sentence following on from the next—place the star command ***CONT** at the beginning of the lexicon. This selects from the choice of sentence structures, places it on the screen, picks another, places it immediately following, and so on. (To the limit of the screen set by ***LIMIT**; if this command is not used, the program fills all available lines on the screen, then halts, waiting for you to click on **More** for the next.)

However, **A.I.Q.** works literally. No spaces will appear between the sentences unless you place them there. For this purpose, you should use the vertical bar, **|**

Our example cat-on-mat sentence from the previous chapter ended up looking like this:

```
[The|A|Your] =3 [=1|=1|=2] sat [=4 ]on [[the|a]|my] =2.
```

To force two spaces to appear between each variable sentence in continuous mode, add two spaces at the end, followed by the vertical bar:

```
[The|A|Your] =3 [=1|=1|=2] sat [=4 ]on [[the|a]|my] =2.  |
```

Think of the bar as a place marker which makes it clear, both to your eye and to **A.I.Q.**, how much space has been allotted at the end of a sentence. Of course it isn’t included in the resulting text.

WRITING A LEXICON FROM SCRATCH

So far we have been working with one of the lexicons supplied with the **A.I.Q.** package. CAT.AIQ and all the other lexicons can be similarly modified: some of them, like PLOT.AIQ and STARS.AIQ have been written with that in mind. But it is also possible to create entirely new lexicons of your own.

Begin from the **A.I.Q.** main screen. On the menu bar click on **File**, then from the pulldown menu that appears click on **New**. A blank text editing screen appears.

Always start simply, test the results as you go along and add complexities in stages. Begin with a single sentence structure, assign key codes to the main constituents of the skeleton, and create word lists correctly coded to those keys. When you create word lists, keep them short until you are sure the syntax of the sentences is correct ... then add as many more as you like. (The more the merrier.)

A.I.Q. allows up to 40 sentence structures, each of up to 255 characters in length. (The **&** and **+** characters—used to join several lines into a single structure, and described in the next chapter—are not included in this total character count).

Any whole sentence structure can *itself* be made variable by enclosing it within pairs of square brackets. The usual rules of likelihood as described above then apply. The only thing to be careful of is to remember to put the same number of opening and closing brackets at each respective end. (I.e., if you put three **[[[** at the beginning, make sure there are three **]]]** at the end!)

Study the list of star commands (next chapter), and make use of them. They extend the power of the program in several ways, and give you a great deal of control over the way the results will appear.

Chapter 6 of this book suggests a few lighthearted ideas for lexicons of your own. It is not difficult to think of more; once you have used the program a few times you will come up with several ideas.

The clue to successful random text lies ironically not so much with the program, but with a grasp of the principles of English. You will regret having dozed off during all that boring school stuff about parsing and syntax ...

5 ● Developing More Complex Lexicons

Variety is the spice of random prose. The more material the program has to work with, the more the results will seem subtle and unpredictable.

In the previous chapters we explained how to take a simple sentence structure, and make it produce random variations on a very straightforward theme. Even at this basic level, **A.I.Q.** will frequently produce results that are unexpected, amusing and sometimes remarkably stimulating.

But **A.I.Q.** is capable of many more levels of randomness, and with practice you will find that there is virtually no limit to its flexibility.

Here now is a rundown of all the extra facilities in the program. Do not attempt to try them all at once. If you have not already worked through Chapters 3 and 4, and tried out the examples, you will probably find this chapter confusing.

FORMAT OF THE LEXICON

Although we have covered this in Chapter 3, it's worth repeating that every lexicon must adhere to a strict format.

It is in three parts:

1. Star commands (see below), which tell the program how to format and display the text.
2. The sentence structure(s).
3. The word lists. These *must* be preceded by a single line which consists of ***Wordlist**. This tells the program that everything following is its "vocabulary" for random selections.

All empty lines will be ignored. You can create empty lines by pressing **Return**, and the use of them will help separate different parts of the lexicon.

Every line must end with a **Return**.

Identifying the Word Lists

When **A.I.Q.** finds the ***Wordlist** command it understands that there are no more sentence structures and that the word lists will follow. Each word list needs a header, with an identifying code before the actual list of words or phrases. This identifying code is used in the sentence structures as the "key" that tells **A.I.Q.** which word list to select from. The insertion code will always cause a search in a word list keyed by the same character. E.g., the insertion code **=G** will always search in the word list headed ***G**.

Either upper- or lower-case characters may be used for the header command (and for its reciprocal insertion command).

Any of the following characters may be used as word list headers and "keys":

Alphabet:	`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
interchangeable with:	`a b c d e f g h i j k l m n o p q r s t u v w x y z`
Numerals:	`0 1 2 3 4 5 6 7 8 9`

And also: `: ; < = > ? @ [ \ ] ^ _`

Only *the character immediately next to the star* is significant. It can be followed by a space, or words, or be a part of a word. E.g.:

```
*N nouns
```

This would be a word list heading used by the insertion code **=N** or **=n**. The word “nouns” is there as an identifying comment only, and has no effect on the program.

```
*N
```

This would work identically, but lacks the comment.

```
*Nouns
```

This also works identically, the header acting both as a code—in its first character, anyway—and a comment. The program reads only the first character after the *, in this case **N** for Nouns.

Because up to 48 wordlists are allowed, you can have several different lists of nouns, verbs, and so on, appropriate to different contexts. Word lists need not be confined to single words: there's nothing to stop you inserting whole phrases which fit the word list's context: in the animals list in the example, you could have added “sabre-toothed tiger”. Even whole sentences can be used in word lists.

STAR COMMANDS

These are the first items that appear in any lexicon. (The ***Wordlist** command is also a star command, but it is unique in being placed *after* the sentence structure.)

In the CAT.AIQ example in the previous chapter, two commands were used: one that displays header text while **A.I.Q.** is running, the other that limits the number of displayed lines to 1.

All star commands may be entered in upper or lower case, or in a mixture of the two. The * must be placed at the extreme left-hand end of the line and each command must appear on a line of its own.

Here is a list of all star commands you may use:

***** [Comment] A star with a space after it: all further text on this line appears only as a comment in the lexicon.

***CAPS** followed by a numeral sets the “capitalization mode”:

*CAPS 0 (zero)	does nothing: all text appears exactly as it has been typed in the sentence structure or word lists.
*CAPS 1	the default setting: the first letter of each expanded sentence outline is capitalized before printing, irrespective of how the word has been typed in the lexicon.
*CAPS 2	capitalizes the first letter of any word produced from the word lists via the = or **%** commands. (See below, for use of the **%** command.)
*CAPS 3	produces all-capitalized output.

You can also force capitals by simply typing in certain words in a list with a capital letter. Or you can use the tilde **~** or <emphasis> commands, below.

***CONT** causes the text output to be continuous ... that is, each new expanded sentence outline starts to appear where the last left off, rather than on a new line. Generally you will want to set ***CONT** when producing chunks of prose, and to leave it off for verse, book titles, short catchphrases, etc.

***END** causes whatever follows (text or variable structures, or a combination of them) to come at the end of the output. If you want this to appear on a new line, make the “new paragraph” command—see =/, below—the first entry in this line. See also ***START**, below, which is used similarly.

***EXCLUSIVE** is used to mark any word list or lists as "non-repeating". When a list is set to be EXCLUSIVE, **A.I.Q.** tries not to repeat any random selection from that list in a screenful of output. (This of course fails once every word in a list has been used in a screen -- at which point the program clears its record of which words have been used and starts again.)

For example: *EXCLUSIVE 1B would mark the ***1** and ***B** word lists (those sought by the insertion codes **=1** and **=B**) as non-repeating.

***EXCLUSIVE ALL** marks *all* your word lists as non-repeating.

***EXCLUSIVE** is used to avoid unwanted repetition of e.g. adjectives: if the ***N** list contains names and the ***A** list adjectives, the command ***EXCLUSIVE A** would prevent **A.I.Q.** from generate (from "=N was =A and =A.") a sentence like "Susan was angry and angry." In the supplied lexicons, LOTTERY uses this feature to avoid repeating its "winning numbers".

***HEADER** (followed by a space, then text of your own) places a message on the header bar of the Results output window. If you do not specify a ***HEADER** line, **A.I.Q.** places the name of the lexicon in this position instead. Note that the text specified in a ***HEADER** line is fixed, not variable: commands like **=1** will be displayed, not acted upon.

***LIMIT** followed by a numeral, sets a limit to the number of sentence outlines which will be processed per screen. See for example STARS.AIQ, where a ***LIMIT 12** command, combined with a ***SEQ** command (see below) ensures that each screen contains all twelve astrological signs in their correct order. ***LIMIT 0** (zero) or ***LIMIT** *without* a numeral sets the limit automatically to the number of sentence structures in the lexicon.

***MAXLINE** can be used to set the last line of the screen window used for output. This is normally set to a conservative 20 but can be much increased. See also ***SEQ**. In the "Configuring lexicon" dialogue, setting ***MAXLINE** to 0 leaves out the command altogether.

***NOCHECK** turns off **A.I.Q.**'s attempts to check and correct the spelling of automatically generated plurals. This is really a relic of early DOS versions that ran on sluggish computers where the extra processing could significantly slow down **A.I.Q.**

***PARA** followed by # or S, or both, formats paragraphs.
***PARA #** ("number") causes each paragraph to be numbered.
***PARA S** ("space out") leaves a blank line after each paragraph.

***PAUSE** followed by a number (of seconds) sets the default "Cycle" time for repeated **A.I.Q.** output, and also starts the cycling automatically without your needing to click on **Cycle** and **Start**. If you omit the number, the default time of 5 seconds is used.

In the "Configuring lexicon" dialogue, setting ***PAUSE** to 0 leaves out the command altogether.

***RESET** deals with a minor side-effect of using too many variables in conjunction with ***SEQ** and/or ***LIMIT** commands. If the sentence structures produce a lot of wildly ambitious random text, you may find that lines occasionally spill over to the next screenful. If this happens, all subsequent screenfuls will also be out of synch. To stop this happening, place the star command ***RESET** at the beginning of the lexicon.

Additionally, if you are using paragraph numbering specified with ***PARA #**, ***RESET** will set the numbering back to 1 for each new screen.

***SEQ** switches off *random* selection of sentences, and forces *sequential* sentences. The sentences appear in the order shown in the lexicon, starting from the first, going through all in turn, then starting again at number 1. This is useful for constructing lexicons which generate text with a particular flow -- like the letter format in the HITECH and HELP lexicons. See also ***RESET**.

***START** causes whatever follows (text or variable structures, or a combination of them) to come at the beginning of the output. If you want this to appear on a separate line, make the "new paragraph" command—see **=/**, below—the last entry in this line. See also ***END**, above, which is used similarly.

***synonyms**. You can design these yourself; see below.

***TAB** followed by a numeral, sets the number of spaces for paragraph indentation. The default is 0 (zero); the maximum is 12.

In the "Configuring lexicon" dialogue, setting **TAB** to 0 leaves out the command altogether.

Setting Star Commands by Menu

All of the above star commands can be entered manually, but you will probably find it easier to use the "Configuring Settings" dialogue box which can be accessed from the lexicon editing screen.

On the menu bar, Click on **Options**, then from the pulldown menu that appears click on **Lexicon Settings**. Alternatively, you can simply click on the **Settings** button. The result in both cases is a dialogue box from which star commands can be automatically inserted into the text of the lexicon.

The dialogue box provides:

- edit boxes for changing the ***HEADER**, ***START**, ***END** and ***EXCLUSIVE** settings;
- radio buttons for selecting the ***CAPS** and ***PARA** settings;
- numerical selectors for ***PAUSE**, ***TAB**, ***MAXLINE** and ***LIMIT**;
- checkboxes for the on-or-off settings ***CONT**, ***SEQ**, ***RESET** and ***NOCHECK**.

The numerical selectors are “spin edit” boxes with tiny “up” and “down” buttons. Click on “up” to increase the value by 1, “down” to decrease it by 1. To make large changes, click on the number box itself: the background changes from white (or the current Windows background colour) to aqua-blue, and the “up” and “down” clicks now add or subtract in steps of 10. Click the box again to restore the white background and change the step value back from 10 to 1.

SYNONYMS

These are commands you can create yourself, and allow you to define the shorthand for variables. The format is: a star, immediately followed by the shorthand word—which can be up to eight letters long—immediately followed by an equals sign. You then type in the variable definition. (If you leave a space, this will be assumed to be part of the definition.) Here's an example:

Suppose you wish to have an article in a sentence, which would be, variably, “the”, “a”, “your” or “my”. (Yes, we know “your” and “my” aren't articles, but....) It uses up a lot of space to type these in every time, so you can nominate a shorthand command instead. Call it “art”, for “article”. The synonym command would therefore look like this:

```
*art=[the|a|your|my]
```

To place these variables in a sentence structure, you may now use the shorthand instead, enclosed within curly brackets: **{art}**.

Another example. Suppose that you *sometimes* like to use an adjective randomly followed by another, for extra emphasis. “The large, hairy lion”, etc ... instead of just “The hairy lion”. Not every time, but maybe on average half the time. If your adjectives are contained in word list ***A** (normally keyed from the sentence structure by **=A**) you could introduce the variable with **=A[, =A]**. Rather than having to remember this every time, you could create a synonym:

```
*adj= =A[, =A]
```

Whenever you want this construction in a sentence, enter **{adj}** instead of all those brackets. Note that there is a space between the command and the synonym ... this will appear in the text. If a synonym is used without having been defined somewhere in the file, the text and its surrounding curly brackets are ignored.

You can use up to 40 synonyms in each lexicon.

OTHER RANDOM CHARACTERS USED IN SENTENCE STRUCTURES

In addition to plain text, word list keys and synonyms, there are several other insertion codes that may be included in sentence structures for different kinds of randomness.

=*	will insert a random letter of the alphabet.
=+	will insert a random consonant (including "Y").
=-	(the hyphen, or minus sign) will insert a random vowel. To give a silly example, an entry in a sentence structure like =+=-=+ will produce a variety of interesting syllables like "Xag" or "Bux" or "Piq", and so on. It will even come up with real words sometimes!
=#	will insert a random numeral from 1 to 9. E.g., to produce a random sterling amount in thousands, enter **£=#,=#=#=#.=#=#**. A dollar amount in hundreds would be **$=#=#=#.=#=#**.
=$	will insert a random numeral from 0 to 9. Used exactly as =#, except that the zero will sometimes appear. In the sterling example above, **£=$,=$=$=$.=$=$** could sometimes produce "£0,256.95" ... so be careful!
=&	will insert whatever random character was *last* generated by one of the above.
=/	commands a new paragraph.
=)	(followed by a digit from 0 to 9) "store" the last randomly generated word list selection. See the next section..
=(	(followed by a digit from 0 to 9) repeat a previously stored word list selection. See the next section.
~	(Used without the = sign.) This forces the character immediately following to be capitalized irrespective of any *CAPS command. It is also used as a "null" character immediately before a numeral, when weighting probabilities in word lists as explained on page 25.
%	Produces *non-random* text, when used instead of the = sign for word lists. In the CAT.AIQ example in the previous chapters, the animal name was keyed by **=1**, producing random selections from word list ***1**. Had this command been **%1** instead, the animal names would have come up in the order of listing, starting at the beginning of list ***1** and returning again to the beginning when the end was reached. This sequence is unaffected by random calls to the same list from other keys in the sentence structure.

The EXAMPLE lexicon shows these commands in action.

MORE ADVANCED LEXICON TECHNIQUES

Storing and Repeating Random Text Items

Suppose you want to repeat a word or phrase which you don't know in advance because **A.I.Q.** has yet to generate it randomly?

The "repeat" feature uses the = sign plus the left or right parentheses. In each case the combination should be followed by a digit from 0 to 9, to distinguish between 10 notional "stores", each able to hold one item. The command **=)0** will *store* the most recent random selection from a wordlist, into store 0. The command **=(0** will *repeat* the contents of store 0 at the current text position. (And similarly for stores 1 to 9.)

Here's a clarifying example. Suppose we want to generate repetitive sentences along the general lines of Joseph Conrad's “The horror! the horror!” One way of doing it would be:

```
The =H=)1! the =(1!
```

... where the *H wordlist contains a suitable selection of appalling concepts, like “horror”, “election result”, “Microsoft Windows Vista”, etc. Whichever is called up by the **=H** is then *stored* by the immediately following **=)1**, in store number 1. The later **=(1** command retrieves a copy of whatever store number 1 might hold. The stored copy will remain available until another **=)1** command overwrites it. Using **=(1** without a previous **=)1** will return no text at all.

Only the random returns from wordlists can be stored in this way; variable text in square brackets won't be. But it's perfectly valid to put something like `[=H|=X|=K]=)5` ... which will store the appropriate return from the *H, *X or *K wordlist, whichever was selected.

The supplied XXX lexicon makes heavy use of this store and repeat feature, which is still no excuse for it; the QUATRAIN verse-writing lexicon uses it to repeat words from particular lists of rhymes, to ensure that the resulting lines always rhyme. The EXAMPLE lexicon also (more simply) shows these commands in action.

Weighting the Chances

In some word lists it will sometimes seem necessary to *increase* the likelihood of certain words being selected. You could do this, of course, by typing in the word many more times ... but this will increase the amount of memory the program uses (and hence slow it down a bit).

You can instead “weight” certain words by directly preceding them with a numeral.

20cat in word list ***1** of our example would give the same advantage to “cat” as typing it in 20 times.

Any numeral between 1 and 99 may be used.

But what if you need to include a number as a word? (E.g., the word “32-bit” in the HITECH lexicon?)

Precede it with a tilde **~** ... when used against a numeral (as opposed to its normal job of capitalizing the next letter) it acts as a “null-effect” character. This won't increase the weighting but will cause “32-bit” to be entered correctly.

Numerals can also be weighted, if the ~ is used. Here are some examples:

`94 London Road`	**A.I.Q.** reads this as “ London Road” (note the space), with a weighting of 94.
`~94 London Road`	**A.I.Q.** reads this as “94 London Road”, with no weighting.
`2~94 London Road`	**A.I.Q.** reads this as “94 London Road”, with a weighting of 2.

When a Sentence Structure is Too Long

As we saw in Chapter 4, even the simplest of sentences rapidly starts spreading across the line as more and more variables are added. What happens if you reach the line-width limit, and part of the structure is “wrapped” to the next line?

Unless you intervene, the program will assume the second part on the new line is a separate sentence structure.

To “join up” the two (or more) parts, insert an ampersand **&** or a plus sign **+** *at the beginning of the new line*. Make sure that you do not add extra spaces or delete intended ones while you do so. (The wordwrap character at the end of the first line will become a **Return** when the document is saved, and the & or + sign counteracts this.)

Emphasizing Text

Any text in the lexicon that is enclosed within angle brackets, **<** and **>**, will be emphasized—shown in capitals—when the program runs. E.g.:

`This is an <Ansible> product` ... comes out as ... `This is an ANSIBLE product.`

If for any reason you wish capitalized text to predominate, include a **<** character in a ***START** string. All text thereafter will be capitalized. Then, if you want some text to appear in ordinary small letters (for negative emphasis, so to speak), enclose it within angle brackets >like this<.

Random emphasis is also possible, using the same syntax as for other variables. For example: This is an [<]Ansible> product. (The **>** has no effect unless the first **<** is there.)

Using angle brackets for highlighting has *no* effect on their use as insertion commands. You may still use **=<** and **%<**, **=>** and **%>**.

Grammar

A.I.Q. is capable of some minor corrections to grammar.

For instance, if you use the word “a” as an article, and the randomly generated word following it happens to begin with a vowel, **A.I.Q.** will quietly change “a” to “an”.

Similarly, if you use the technique of forcing two words together (as we have done in FANTASY), and those two words end and begin with the same letter, a hyphen will be inserted. E.g., “Darkking” becomes “Dark-King”.

Plurals are made as logical as possible. E.g., “**=N**s”, selecting nouns from word list ***N** would pluralize “cat” as “cats”. Words ending with “x” have the necessary “e” inserted automatically: the plural of “box” becomes “boxes”.

In the same way, “-chs” becomes “-ches”; “-fes” becomes “-ves”; “-lfs” becomes “-lves”; “-rfs” becomes “-rves”; “-shs” becomes “-shes”; “-sss” becomes “-sses”; “-ys” becomes “-ies”; “-zs” becomes “-zes”.

There *are* limits: the plural of “kibbutz” becomes “kibbutzes”, not “kibbutzim”. We never claimed that **A.I.Q.** was a kosher program.

6 • More Ideas for A.I.Q.

We hope you will use **A.I.Q.** for diversion and amusement, and, having learnt the dreadful lesson of FUSTIAN.AIQ, not try to write **King Lear** with it. What seriousness lies in **A.I.Q.** is the seriousness of language, and the subtleties and secondary meanings that can arise when words and ideas are placed unexpectedly together. We have tried to demonstrate this with the lexicons we have supplied, but we see these as just a start. Learn the program, add to our word lists, or compile your own from scratch. As soon as you see your own input arising magically from the limbo of the computer's memory, you'll begin to sense the possibilities. The only real limitations on **A.I.Q.** are the limitations of language.

To get you going, here are a few silly ideas for lexicons you can try producing yourself:

INSTRUCTION MANUAL. A beginner's guide to servicing a car, building a concrete shed, feeding a pet rabbit, programming a video recorder, etc.

MOVIE GREATS. Films no one has ever seen, directed by nobodies, starring actors no one has heard of.

STRAIGHT FROM THE HEART. The world's longest love-letter, full of embarrassing details. Keep it clean!

```
"I held your =G, while you breathed =F =D in my =R. My =X pounded."
```

THE VOICE OF WESTMINSTER. Write down all the parliamentary phrases you can imagine and have **A.I.Q** combine them into endless political woffle. Don't forget to program in spontaneous asides and replies to interruptions. (`"My right hon. friend on the opposite bench may well fidget/close his eyes/wince/jeer/run headlong from the Chamber...."`) Save yourself and all your friends the trouble of listening to Question Time on TV....

LIMERICKS. Advanced **A.I.Q.** students only:

```
There was a =Y =W of =Q
Who =D a =G of =H =X,
   [He|She] =V a =H =L
   And =Z a =H =M
Because =C a =G =N and =A!
```

7 • Windows Menus and Commands

A.I.Q. uses several common Windows features, which we assume are already familiar to you. They are described in the Windows manual, if not. Here is a note of the Windows commands you can use which have special effect in **A.I.Q.**:

Main A.I.Q. Screen Menu Bar

File	New	• Opens an untitled new lexicon edit screen.
	Open	• Loads an **A.I.Q.** lexicon from disk (Open dialogue box).
	Save *	• Save current lexicon to disk.
	Save As *	• Save current lexicon under a new name.
	Run A.I.Q. *	• Run **A.I.Q.** to produce random text.
	Close *	• Close the current lexicon.
	Exit	• Exit from the **A.I.Q.** program.
Edit *		• All the standard Windows clipboard commands: Cut, Copy, Paste, Delete, Undo and Select All, plus Find and Repeat Find for lexicon navigation.
Character	Font * †	• Change screen font of lexicon text.
Options	Lexicon Settings	• Opens “configure lexicon” dialogue box.
	Minimize on Open †	• ✓ = newly opened lexicon appears as an icon only (with no ✓, the lexicon edit window automatically opens).
	Hint Help †	• ✓ = Hint Help on ... provides reminders of what toolbar and other buttons do, when the cursor is held briefly over them.
Window		• Standard Windows options.
Help		• Standard Windows options ... always consult the Help Contents for new information which may have been added since this manual!

[* Disabled or invisible when no lexicon is being edited ... † Setting saved in AIQ.INI]

ToolBar or Button Bar

Open	• Loads an **A.I.Q.** lexicon from disk (dialogue; same as File | Open).
A.I.Q.*	• **Run A.I.Q.** to produce random text (same as File | Run).
Settings*	• Opens “configure lexicon” dialogue box (same as Options | Lexicon Settings).
Save*	• Save the current lexicon to disk (same as File | Save).
Close*	• Close the current lexicon (same as File | Close).
Exit	• Close down the **A.I.Q.** program.

[* Disabled when no lexicon is being edited]

Lexicon Editor Pop-Up Menu (right mouse click)

= Codes	• Invokes the “Insert into Lexicon” dialogue, from which special commands and any existing wordlist commands can be entered automatically. A radio button selector allows switching between commands like =1 and %1 for, respectively, random and sequential selection from word list 1. The box also displays the available (that is, not yet used) wordlist key characters. When working with dozens of wordlists, this list box is a valuable memory aid, provided you have annotated your wordlist header lines with identifying comments. For example, a header line “*1 animals” will produce, here, the line “=1 animals”—which if selected will insert =1 into the lexicon.
New word list	• Automatically inserts a new (not previously used) word list header line, like “=5 ” ... ready for you to add an identifying comment. Disabled unless there is a ***WORDLIST** command and the cursor is positioned after this in the lexicon editor.
***WORDLIST**	• Inserts a ***WORDLIST** star command into the lexicon. Disabled if one already exists.

Random Text Results Screen Menu Bar

File	Save	• Save current text to .TXT file (the file name is automatically generated from the lexicon name, so CAT.AIQ gives CAT.TXT).
	Exit	• Close the random text display window.
More		• Generate another screen of random text.
Cycle	Start/Stop	• Begin and end regular auto-generation of text screens at default intervals of 5 seconds. (A *PAUSE lexicon command can set a different interval and will also start auto-generation without your needing to use the Cycle options.)
	1 sec ... 60 sec	• Select a preset auto-generation delay, and begin.
Edit		• All the standard Windows clipboard commands: Cut, Copy, Paste, Delete, Undo and Select All.
Options	Font †	• Change screen font of results display text.
Exit		• Close random text display and return to the main **A.I.Q.** window.

[† Setting saved in AIQ.INI]

AIQ.INI Configuration File

This file is created by **A.I.Q.** and saved in its working directory. It “remembers” various program settings, including the Font and ✓ menu selections noted above. Also saved in AIQ.INI: the directory path used for lexicon files and various items of window information, including the window status (maximized or not) of the main, lexicon-editor and results-display windows.

8 • AIQ.EXE for DOS

This DOS version of **A.I.Q.** is supplied on the same disk as the Windows program with which this manual has been most concerned. Its general operation is closely similar, and in some ways more streamlined—not to say simple-minded. When run with the **AIQ** command (typed at the DOS prompt) it immediately displays a directory of lexicons and, when one is selected, immediately runs it to produce a screenful of random text.

At this point, there are three keyboard options:

- **F** for **File** instructs the program to begin saving its random output to a disk file with the .TXT extension.
- **C** for **Continuous** generates repeated text at specified time intervals, like the Cycle option in AIQ for Window.
- **Esc** exits from the current lexicon and redisplays the directory.

HOW THE DOS AND WINDOWS VERSIONS DIFFER

- **A.I.Q.** for DOS has no built-in lexicon editor (DOS EDIT is perfectly adequate), and the lexicon file names end in .DAT rather than .AIQ. In fact the lexicon format is almost identical, but certain characters—accented letters like é, or symbols like £ or ½—have different numerical values in the DOS and Windows character set: a DOS £ sign comes out wrong in Windows. Hence the segregation of .DAT and .AIQ files. Incidentally, AIQ for Windows will automatically convert the character set of a .DAT file upon loading.
- One extra star command is found in **A.I.Q.** for DOS only: ***FILE**, which causes all random output to be written automatically to disk, with the text from CAT.DAT going to CAT.TXT and so on.
- Emphasis using angle brackets, as in `An <Ansible Information> Program`, gives bold or bright text instead of capitals on the DOS screen: `An` **`Ansible Information`** `Program`.

DOS COLOUR CONFIGURATION

CFG.EXE is Ansible's all-purpose utility for setting screen colours (and/or mono appearance) in our IBM programs. It can be run from the DOS prompt with the command **CFG**.

The CFG command screen shows a list of numbered options—more in fact than are required by **A.I.Q.** itself. You will need only the first four, numbered computer-fashion from 0 to 3. An option can be selected by typing its number, or by moving the triangular markers to its position with the up- and down-arrow keys and then pressing **Return**. (Just to provide an embarrassment of options, numbers 1 to 3 can also be selected with **F1** to **F3**.)

Option 1 allows you to change the way **A.I.Q.** shows “normal” text (the bulk of its output), option 2 controls “bold” (e.g. the emphasized letters commanded by **`<angle brackets>`**) and option 3 controls “reverse video” (e.g. the highlighted end-of-screen message bar).

After selecting option 1, 2 or 3, go on as follows....

On true colour monitors, entering one of the letters A to H will immediately select a foreground colour, and I to P a background colour. You are free to experiment. The available colours are shown near the bottom of the screen. On the whole, **A.I.Q.** looks best with the same background colour for normal and bold text.

For grey-scale monitors like the long obsolete VGA Mono display, the CFG screen will still say “COLOURS” at top right (since the hardware is colour-compatible and sending colour signals) but the letter options give only white, black and shades of grey.

For true monochrome hardware where the screen says MONO at top right, only five options other than total invisibility are available (normal, bright, underlined, bright underlined, and inverse), and these are chosen by entering A to F.

On all monitors, you may also cause the text to blink horribly. This option is turned on, and hastily off again, with Z.

On all monitors, you should when satisfied “accept” the current text appearance with **Return**. The default for the selected option can be restored by first pressing **Tab**. Pressing **Tab** when not currently editing one of the text options will restore *all* the colour defaults.

Having commanded and “accepted” each desired change, you can leave the program either by selecting option 0 (zero) to save the changed settings, or by pressing **F10** or **Esc** to leave the program without saving any changes.

CFG.EXE's settings are saved in a little file called ANSIBLE.CFG, which **A.I.Q.** for DOS will read (if it's present) whenever it loads.

The End