Capture and Storage of Respiratory Test Data

by

Cheah Ui Poh

as partial requirement for the degree of

Master of Philosophy

at

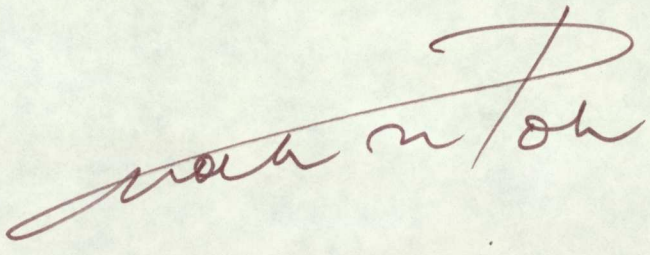
The Department of Production Technology and Production Management,
The University of Aston in Birmingham

on

May 1982

Declaration

Unless specifically described, the work described in this thesis has not been done in collaboration. This work has not been submitted for any other academic award at this or any other educational establishment.



Capture and Storage of Respiratory Test Data

Cheah Ui Poh Master of Philosophy, 1982

Summary

This thesis is concerned with the online collection and storage of respiratory data. Initially, a feasibility study was done to see if it was possible to use a 'microcomputer' to perform the data collection at the required speeds. The advantages of using a 'microcomputer' are two-fold. First, the system would be portable in that it can be taken to the patient rather than vice-versa. Second, it would be easier to use a dedicated microcomputer rather than rescheduling a time-sharing minicomputer to perform the data collection.

On completion of the feasibility study, it was found that there was a vast amount of data to be stored. It was calculated that a 20 megabyte disk could only store the results of 35 days of testing! An attempt was made to reduce the amount of data in such a way that it could be reproduced in its entirety when required.

Since the data was unavailable during the data reduction exercise, it was typed in manually. However, as this was both time consuming and error prone, an automatic process was sought. This involved linking the 'microcomputer' to the minicomputer and transferring the data directly.

Finally, an information management system was designed so that data with certain characteristics could be reproduced when it was required.

Keywords: analog to digital conversion, curve fitting, data transmission, information management systems

LIST OF CONTENTS

LIST OF FIGURES

List of Acronyms

3D	Digital Design and Development Limited
ADC	Analog to Digital Converter
ANSI	American National Standards Institute
APC	All Points Curve
ASCII	American Standard Code for Information Interchange
CBM	Commodore Business Machines
CCITT	International Consultative Committee for Telephone and Telegraph [the letters are jumbled because it is a French abbreviation]
CIU	Clinical Investigation Unit, Dudley Road Hospital, Birmingham
CMC	Connecticut Microcomputers
DAV	Data Valid
DCC	Data Collection Computer
DCE	Data Collection Equipment
DEC	Digital Equipment Corporation
DG	Data General Corporation
DTE	Data Transmission Equipment
EIA	Electronic Industries Association
eq	Equal to
ge	Greater than or equal to
GPIB	General Purpose Interface Bus
gt	Greater than
HC	Host Computer
HPIB	Hewlett-Packard Interface Bus

IEEE	Institute of Electronic and Electrical Engineers
IEC	International Electro-technical Commission
IN	Infix Notation
ior	Inclusive OR
IRC	Information Retrieval Computer
ISAM	Indexed Sequential Access Method
ISF	Indexed Sequential File
JCL	Job Control Language
K	Computer Kilo = 1024 [2^{10}]
k	Metric Kilo = 1000
le	Less than or equal to
LRU	Least Recently Used
lt	Less than
MS	Menu System
MUMPS	Massachusettes General Hospital Utility Multi-Programming System
NDAC	Not Data Accepted
ne	Not equal to
ng	Not greater than
nl	Not less than
NRFD	Not Ready For Data
OS	Operating System
PCC	Pressure-Concentration Curve
PET	Personal Electronic Transactor
PKF	Primary Key File
PLFBTE	Pulmonary Lung Function Breath Test Equipment
PN	Polish Notation

PUP	Parallel User Port
QL	Query Language
RPN	Reverse Polish Notation
SD	Significant Digits
SDF	Source Details File
TRF	Test Results File
VDU	Visual Display Unit
xor	Exclusive OR

CHAPTER 1: INTRODUCTION

I have a dream ...
- Dr Martin Luther King

1.1 The Current State of Affairs

There are, at present, several types of respiratory lung function testing apparatus on the market. Although some are more expensive and more sophisticated than others, they all use the same techniques for measuring the various coefficients. Most of them place a large emphasis on the operators for the interpretation of their results.

With some equipment, the operators are expected to translate the multitude of results generated to something understood by the physicians [Hancox [1980]]. These results are normally obtained by taking values off a graph produced by a plotter. Measuring plotter output is tedious and often prone to error. It also causes problems in storage and thus is always discarded after the relavant measurements have been taken. There is also the problem of knowing what tests to perform.

Sometimes, the 'discarded' graphs are collected by research workers so that they can prove or disprove hypotheses

emphirically. Sometimes they are used for statistics. It normally takes a very long time to collect the required data; for instance, in a typical three year Ph.D., one year could be spent collecting the data.

From the patient's point of view, these tests are a chore and often difficult to perform. Some patients just cannot perform the test and some insist that they cannot do it. Since the machines are generally built to suit adults, it may be difficult to use for children. They require smaller mouthpieces and are frequently apprehensive of the 'fearsome looking' equipment.

1.2 The Requirements

It is quite obvious that the operators' jobs would be simplified if they could spend more time with the patients while the equipment monitored itself. It would also help if there was some way of telling the operator the tests which should be performed given the results of the previous test.

On the research side, the information could be automatically accumulated for further reference [ie data bank]. This would best be kept in its raw form and processed as and when it it required. It could be kept in a processed form but it should be possible to obtain the raw data from this processed form lest a different form of processing is required.

By marrying the above requirements, the monitoring equipment could be directly connected to a data bank, thereby reducing errors inherent in manual transcription.

1.3 Fulfilling the Requirements

1.3.1 Automatic Monitoring

An approach that could be used for automatic monitoring of the equipment would be to connect it directly to a computer. The computer could be programmed to monitor and collect data from the equipment. When such information had been verified to be correct, it could be transferred to the data bank.

1.3.2 The Data Bank

There are several techniques for building a data bank. Avison [1981] has identified three common techniques used in scientific work:

- Data Collection
- Entity Relationship
- Data Abstraction

Briefly, in the Data Collection Technique, all the information which is deemed useful is accumulated. The information is then structured as required.

In the Entity-Relationship technique proposed by Chen [1977], the entities ['things'] of interest are first identified. The relationships are then formed between these entities, and the attributes are identified. Once this has been done, the formats can be designed.

The Data Abstraction Technique developed by Smith and Smith [1977] views the data model as an abstraction of the real world. All potential users of the model are surveyed to determine their information needs. These requirements are then aggregated and generalised [that is, common information needs are grouped together], to build an information structure for each user. The information structures are then integrated to form an overall structure.

The technique which will be used is a mixture of the first two described. The requirements of future users of the system are unknown but relationships can be set up amongst the data collected. This, however, will not be a well defined collection; any form of data which may be useful will be collected.

1.4 The Approach

The solutions to the problems will be attempted in the following phases:

- Identification and partial implementation of a method of collecting information from the Pulmonary Lung Function Breath Test Equipment using a computer.
- Identification and solution of problems in the previous phase until the data is in such a state that it can be stored easily.
- Design of a data bank and suite of programs which will extract information which users will require.

End of Chapter

CHAPTER 2: CAPTURING THE DATA

Anything can be made to work if you fiddle with it long enough - Wyszkowski's Second Law

2.1 Introduction

The objective of this 'sub-project' is to assess the feasibility of using a microcomputer [term explained in section 2.2.1] for the automatic collection of data from Pulmonary Lung Function Breath Test Equimpment [PLFBTE]. A method which is often used for automatic data collection is the linkage of the PLFBTE to an Analog to Digital Converter [ADC]. In this chapter, the hardware requirements, software interfacing and testing of such a system will be examined.

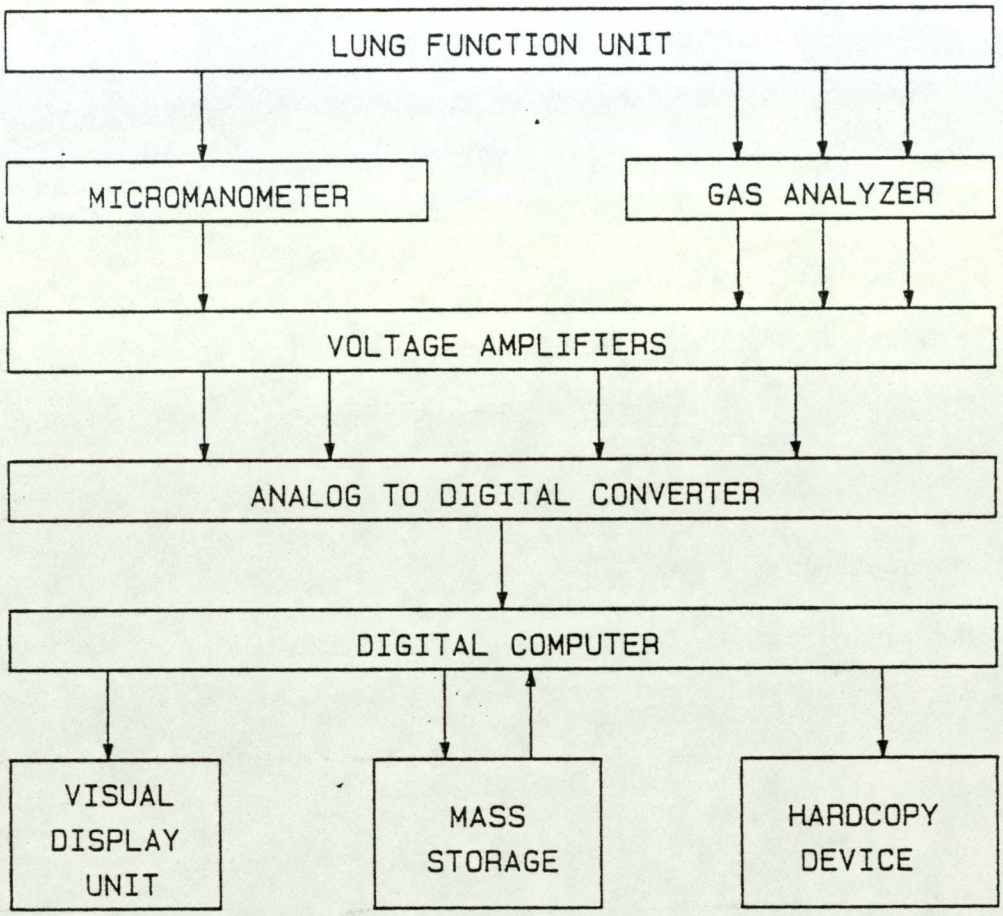


Figure 2a: Block Diagram illustrating the flow of information from the Lung Function Unit to the Digital Computer via transducers.

2.2 Hardware requirements

The hardware requirements are listed as follows:

- a dedicated small computer
- a printer with possibly a graphics option
- a facility for reasonable mass storage or communications
- an analog to digital converter which is compatible with the computer and has suitable sampling characteristics

Figure 2a illustrates the flow of information from the PLFBTE to the computer.

2.2.1 Types of Computers

A computer is basically a collection of logic devices. Electronic computers were originally divided into two classes: mainframes and minicomputers. The mainframes were capable of handling a number of jobs concurrently whilst the minicomputers where only capable of handling one job at a time. Advances in electronics brought about the microcomputer, originally defined as a variety of logic devices implemented on a single chip. The term 'microprocessor' was originally coined to reflect the limited functions of logic devices as compared with a microcomputer.

With the introduction of multi-tasking minicomputers and further advances in electronics, the above definitions have been obscured. The term microcomputer has almost lost its original meaning. Two reasons can be attributed to this:

- An increasing number of microcomputers are single chip implementations of existing minicomputers
- Some manufacturers build their own computers out of microcomputers and microprocessors [or support chips]. These machines are capable of handling programs like the old minicomputers but for some odd reason [like sales strategy] are called microcomputers!

Thus, a computer is now called a mainframe, a minicomputer, a midicomputer, a super-mini or a microcomputer simply because that is what the manufacturer has decided to call it. It bears no real relation to the ability or speed of the machine. In the text that follows, it will be assumed that a microcomputer is cheaper than a minicomputer, which in turn is cheaper than a mainframe.

2.2.2 The Dedicated Small Computer

Since the system is to be operated 'on-line', the computer must be able to devote its full attention to the data collection. As a result, it has to be either a single user machine or a machine where it is possible to switch from a single to a multi user

context and vice-versa. Since the data collection computer [DCC] must be a single user machine, the most economical choice would be a microcomputer. There are, however, a very wide range of such machines to choose from and they come in all shapes and sizes. To make life easier, some criteria were drawn up:

- The machine should be portable in that it can be mounted on a trolley along with the PLFBTE and moved around.
- The internal processor [microcomputer?] should be fairly popular; if there is a hardware fault, at least some spares will be available. There is also the advantage of 'off the shelf' software and hardware if it is required.
- The machine should have a facility for analog to digital conversion.

The popular processors around at that time were Zilog's Z80 and MOS Technology's MCS 6502. Since the author was more familiar with Motorola's MC 6802, an elder cousin of the 6502, it was decided that a 6502 based machine would be used. There were very few 6502 based machines which were fully assembled; most of them came in kit form. Of the machines which were fully assembled, only two machines satisfied all of the above conditions: the Apple II and the Commodore PET.

It is often debated in 'Home Computer' magazines as to which is the better of the two. The Apple is a better machine internally; it is more flexible, well designed, has better documentation but has a very flimsy frame. The disk drives, display unit etc are all separate units and are connected by means of the Apple's 'S-100 Bus' [IEEE-696 Standard]. It does have the added advantage of high resolution colour graphics.

Until lately, the PET has had poor documentation riddled with errors. It is a better machine in terms of robustness and editing facilities. The PET has a built in display unit and communicates via the GPIB Port [see Chapter 4], its Parallel User Port and two Cassette Ports. When machines are almost equal, the deciding factor is often the price. Since the PET was cheaper overall, especially when floppy disk storage was also considered, it was favoured.

2.2.3 Operation of an Analog to Digital Converter

An ADC basically converts an analog voltage to a set of binary digits. There are various techniques used in ADC's, which include successive approximation, duual slope, ramp integration and others. Each technique compares the input voltage with a voltage generated by the ADC until the two are equal.

Successive approximation 'hunts' by binary search until the correct level is reached, while slope techniques use a ramp and a comparator. When the appropriate level is reached, the binary value used to generate the voltage may then be read from the ADC.

Often, to avoid the 'droop' or rapid change problems encountered when an analog to digital conversion time is long compared with the rate of change of the signal, a 'sample and hold' mechnaism is required to prevent changes occuring (which may be greater than $\pm$0.5 least significant bit) during the conversion time.

Of the ADCs available for the PET, the range produced by CMC was chosen and these have the following facilities:

- successive approximation
- 16 input channels
- 8 bit resolution
- input voltages of 0 to 5.12 volts
- conversion time of 100 microseconds
- absolte maximum error of 0.7%

2.2.4 The Analog to Digital Converter

The ADC chosen was the AIM161: an 8-bit, 16 channel ADC manufactured by Connecticut Microcomputers [CMC]. The reasons for the choice follow:

- At that time, there were only two ADCs available for the PET: one by CMC and another by Digital Design and Development [3D]. The one by 3D had exactly the same features as the one by CMC but it cost twice as much!
- The AIM161 did not have to be connected directly to the GPIB and so did not require any form of handshaking.
- The conversion time was about 100 microseconds. This implies that the number of samples per second could be upwards of 10k. Since 500 samples per second was more than adequate, there would be no problems as far as settling time etc. were concerned.
- There were no 12 bit ADCs for the PET in the market at that time and a lot of time would be wasted if one had to be constructed and debugged. Moreover, it was uncertain whether this idea would work or not so the less money spent the better.

2.3 System Testing: Theory and Practice

Before any systems software can be written, the procedures for checking it should be programmed. In Analog to Digital Conversion, there are two basic entities that have to be tested:

- the accuracy of the system
- the speed of the system

2.3.1 The Accuracy

The technique for checking out the accuracy of the ADC is simple. Since the relationship between the input voltage and the output value is linear, linear interpolationcan be used. If an input voltage of IV volts is applied then

$$(OV - O_l)/(O_h - O_l) = (IV - I_l)/(I_h - I_l)$$

where O = output value

I = input voltage

h = maximum [high] value

l = minimum [low] value

For the AIM 161, this expression reduces to

$$(OV - 0)/(255 - 0) = (IV - 0)/(5.10 - 0)$$

$$OV = IV * 255 / 5.10$$

$$OV = IV * 50$$

Since the output value must be an integer,

$$OV = \text{int}(IV * 50) \pm 1$$

Thus, for an input value of 3 volts,

$$OV = \text{int}(3 * 50) \pm 1$$

$$= 149 \text{ or } 151 \text{ or } 150$$

In practice, this can be done by connecting the ADC to either a reference voltage cell or a voltage calibrator.

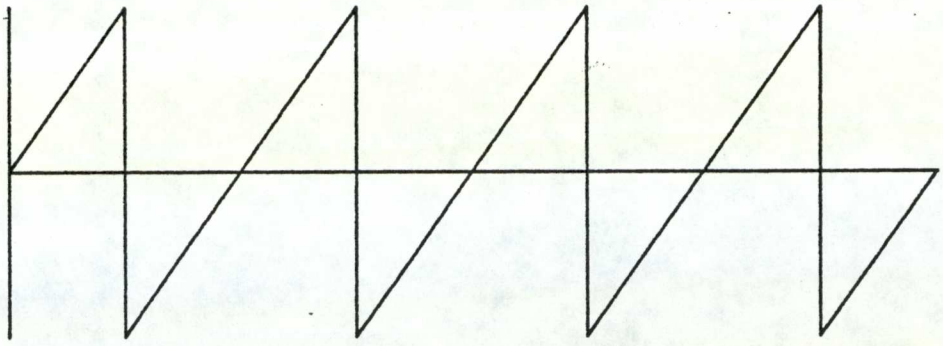
2.3.2 The Speed of the System

The theory behind the evaluation of the sampling rate of the system is fairly simple:

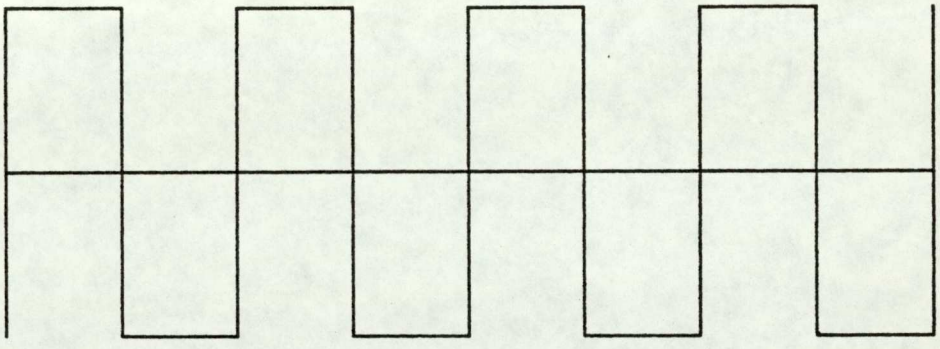
- count the number of samples, ns, obtained in a known period of time, dt [in seconds]
- divide ns by dt to obtain the frequency [in hertz]

In practice, this can be done with the aid of a Signal Generator [SG] or an oscillator. Most SGs provide the wave forms shown in Figure 2b [though the saw-tooth is quite rare].

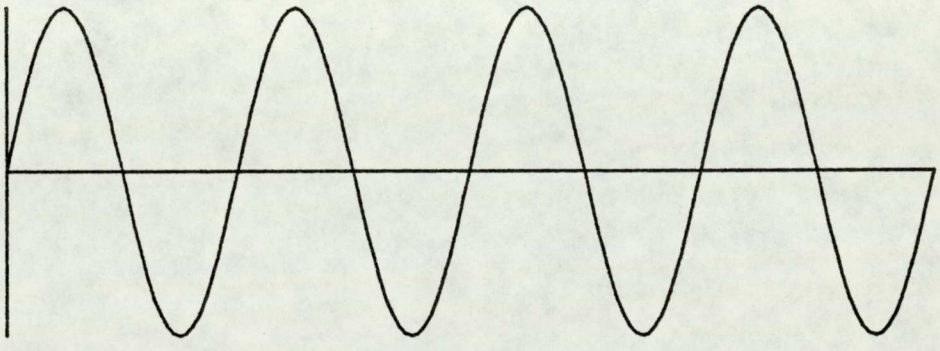
The waveform which should be used is one which clearly shows the positions where a cycle starts and ends [since one cycle is a known period of time]. By observation, any waveform could be used; but if the system is to be automatic, then either the square wave or the saw-tooth should be used since the termination of a cycle on both these waveforms is distinct from the voltage change. Due to the rarity of the saw-tooth waveform on SGs, the evaluation of the sampling frequency with the aid of a square wave.



Saw Tooth



Square Wave



Sinusoidal

Figure 2b: Waveforms available on Oscillators and Signal Generators

On the square wave, the change in voltage is distinct after half a cycle. Using the notation described,

sf = ns / dt

Thus, if the SG is set to generate a square wave in 200 millisecond cycles and 368 samples are obtained in half a cycle,

dt = 200E-3 / 2
   = 100E-3 seconds
sf = 368 / 100E-3
   = 3680 Hz

2.4 The Software Interface

2.4.1 Initial Testing of the ADC

The ADC was tested using the test programs supplied by CMC. The first program sampled each channel in turn with nothing connected to the ports. These gave wild readings between 80 and 255. The second program was similar but this time a 3 volt battery was connected. These gave readings between 149 and 151 which indicated that the ADC was working.

A short program was then written in Basic to test the speed of the software. The program was executed for a minute and the number of samples obtained was then divided by 60. The result was quite depressing: only 5 readings per second!

The above program was then re-written in Assembler. In a short test of one minute, the PET stopped working! The reason for this was obvious; if CMC's claim of a conversion time of 100 microseconds could be assumed and another 100 microseconds could be assumed for the housekeeping software, it would take 200 microseconds for each sample. This implied that in one minute 300k samples would be obtained. Since the data was stored, the program was overwriting itself, hence the reason for the crash!

Two problems arose:

- How the samples would be taken
- How the program would 'know' when it ought to sample

2.4.2 Sampling Techniques

Nyquist's Theorem dictates that to preserve an analog signal completely, the sampling rate should be twice as fast as the sample frequency present in the signal. In practice, sampling rates are more than twice this rate. The problem is how a set of samples should be taken. The data can be sampled in one of three ways:

- at even intervals
- alternately with reference to a clock pulse on one channel
- in bursts

To sample at even intervals, an external clock has to be connected to the PET at some point other than the ADC. Using the second method, the clock is connected to one of the channels of the ADC. Both these techniques will require interpolation or extrapolation of the data at the end of the sampling period so that the samples look as if they were taken at the same point in time.

Sampling in bursts will also require a clock of some sort. It can be used if the conversion time is negligible compared to the time interval between samples. It has the advantage in that the data need not be interpolated or extrapolated.

2.4.3 The Clock

There are two internal and one external clocks on the PET. The internal clocks work at 16.6 milliseconds and 26.7 microseconds while the external one works at 20 nanosecond intervals. It is not safe to use either of the internal clocks as they can be unknowingly locked out by software.

The external clock sends pulses to the internal clock software and other chips on the PET. To use it would require detailed knowledge of PET hardware: information which Commodore was not willing to part with. Additional hardware could be included to give the clock pulse but this was undesirable as routines would have to be written to specially poll a line for the pulse.

On re-examination the resources available, it was found that the 6502 on the PET used the external clock with pulses of one microsecond for its cycle time. This meant that if interrupts are supressed, the cycle time would be exactly one microsecond.

2.4.4 Evaluation of the Delay Time

The following piece of code was written as a delay between sampling periods:

```
Total Cycles
      3           LDX N        ; number of times to loop

    N*2    WAIT   DEX          ; decrement counter
(N-1)*2+1         BNE WAIT     ; loop if X not zero

      4    EOC    LDA USRPRT   ; in case ADC not ready
      2           BPL EOC      ; wait till end of conv.
```

Assuming the second loop is only executed once, the length of this delay would be 13+(N-1)*5 microseconds. In order to obtain a delay of 200 microseconds,

$$13 + (N - 1) * 5 = 200$$

$$N = 38 \text{ approx.}$$

The numbers in the pieces arithmetic which follow will be rounded since it is not possible to account for fractions of a cycle. A delay time of 200 microseconds would theoretically imply a sampling frequency of 5 kHz. Unfortunately, a sampling frequency of 3.68 kHz was obtained. The discrepency between the theoretical and the actual result was obvious: the code which handled the rest of the 'conversation' between the ADC and the PET had been excluded from the calculations.

This discrepency was put right by the following piece of arithmetic:

$$CT + (N - 1) * 5 = 1000 / 3.68$$

Since $N = 38$, $CT = 1000 / 3.68 - (38 - 1) * 5$

$$CT = 87 \text{ appx.}$$

The general equation for evaluating N would therefore be

$$DELAY = 87 + (N - 1) * 5$$

Therefore $N = (DELAY - 87) / 5 + 1$

The delay of 200 microseconds could be effected by

$$N = (200 - 87) / 5 + 1$$

$$= 24$$

With this modification, the sampling frequency was approximately 5 kHz.

2.4.5 Altering the Circuitry

Although the AIM161 did not require the GPIB handshake, it used some lines from the GPIB port. The ADC kit, as it came from the factory was wired so that the signal to start strobing came from a pin of the GPIB port. If the PET had other devices attached to its GPIB port, they could activate [for instance, the printer could start printing gibberish without being 'asked' to]. These unwanted actions could be avoided by altering the printed circuit boards.

The prints on the circuit board indicated the following:

A = pin D of the PET GPIB port

B = pin M of the PET User port

C = line to stop strobing

The manual said that the foil line between A and C should be cut and a jumper should be soldered between A and B. This would send the signals from the PUP to the GPIB! The jumper should have been soldered between B and C, not between A and B as the manual suggested. Unfortunately, this was not realized until very much later and a lot of time was wasted in attempting to debug fully working programs. So much for Cahn's Axiom*!

---

* When all else fails read the instructions.

2.5 General Software

2.5.1 Calibration of Channels

Before any data can be captured, the channels have to be calibrated. The procedure is fairly simple; the operator provides two known quantities while the computer records their corresponding voltages. Since the relationship between the input quantity and its corresponding input voltage is linear, linear interpolation may be used to evaluate the actual value which any other input voltage represented. This would imply

$$(Q_i - Q_l) / (V_i - V_l) = (Q_h - Q_l) / (V_h - V_l)$$

where Q is the actual quantity, V its corresponding voltage, h the high value and l the low value. This derives

$$Q_i = Q_l + (V_i - V_l) * (Q_h - Q_l) / (V_h - V_l)$$

If $S = (Q_h - Q_l) / (V_h - V_l)$

then $Q_i = Q_l + (V_i - V_l) * S$

The values of Q_l, V_l and S will be recorded when saving the results of the conversion so that the results can be converted into their actual quantities when the analysis is performed.

2.5.2 Displaying the Results

2.5.2.1 As provided by the Program

The display is given in two forms; these are user selectable

- in X-Y graphs
- in tables

The operator is allowed to select the axes on the X-Y graphs. Due to the limited graphics capabilities of the PET, the graphs produced are coarse line printer or screen graphs. This should be adequate since the graphs were provided to show the relationships between the input channels. They are useful for detecting air leaks and other such disasters.

For quantitave analysis, the program provides the operator with user selectable columns for tabular output. Up to eight columns are allowed on the printer. If a hardcopy is not required then only four columns are allowed because of the 40 column restriction of the PET's screen. The operator is also given the choice of the amount of output required. For instance, every fifth sample could be selected.

THE FOUR VARIABLES ARE ASSIGNED AS SHOWN, AND THE POSITION/SHAPE OF THE ELLIPSE IS DETERMINED BY P1/P2, WITH ITS POSITION BY P3/P4.

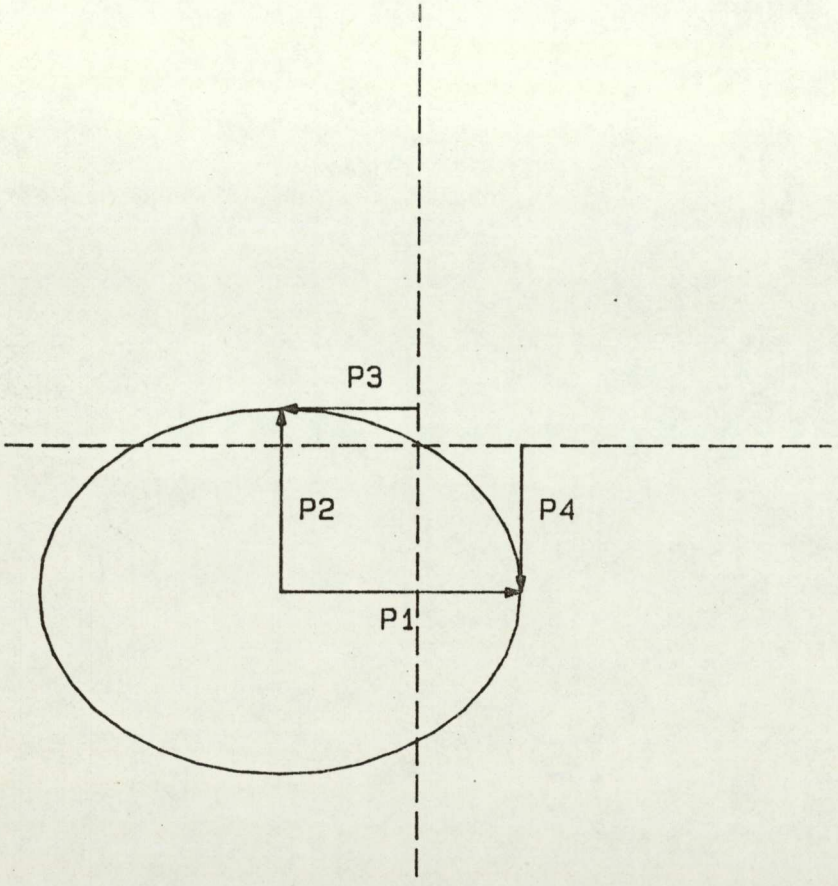


Figure 2c: A method of showing the relationships between four variables in two dimensions.

2.5.2.2 Other Forms of Output

Since the PET has limited graphics capabilities, the types of output it can produce are restricted. If a computer with high resolution graphics facilities [eg Sharp MZ80B] were used, then some other techniques of displaying the data could be implemented. Figure 2c shows a technique of mapping four parameters into two dimensional space. This is similar to a technique to Best [1980] for recording the quantities of various compounds in urea.

2.5.3 Saving the Results

Once the data had been captured, it had to be transferred to a less volatile medium, namely diskettes. On the PET, it is stored as an integer array so it could be transferred in its character format for integers, the machine format of bytes or as fully converted floating point numbers. The least space consuming format was the machine format. This, however, meant that the data used for interpolation also had to be stored. For consistency, the internal format was used.

For each channel, the order in which the data was stored is given as follows:

Lower value of quantity [Q_l]

Lower corresponding voltage [V_l]

Higher value of quantity [Q_h]

Higher corresponding voltage [V_h]

time period

number of points

integer array for that channel

2.6 Conclusion

Since a sampling rate of only 250 Hz [Hancox [1980]] is required and a sampling rate of up to 5 kHz could be obtained, it can be concluded that it is feasible to use a 'microcomputer' to aquire data from the PLFBTE and store it temporarily. Some extensions to the project will be highlighted in Section 2.7.

2.7 Further Work

There are two areas in which further work could be done:

- Linking the PET to the PLFBTE
- Getting the PET to control the PLFBTE

2.7.1 Linking the PET to the PLFBTE

The following steps would have to be taken to link the PET to the PLFBTE:

- Selection of suitable transducers to convert the output values of the PLFBTE to volts [eg a micromanometer could be used as a pressure transducer].
- Selection of suitable voltage amplifiers to bring the output voltage of the transducer to the range required by the ADC.
- Alteration of certain sections of the program so that it 'knows' when it ought to start recording the data. At the moment, recording starts as soon as the 'space bar' is depressed.

2.7.2 Controlling the PLFBTE and experiments

Using a Digital to Analog converter or the Parallel User Port, the PET could be used to contrl the 'Midhurst Valve' and the incoming gases. This would be useful for the operators in that they could spend more time with the patient rather than dividing it between the patient and the equipment.

Work is currently being done at the CIU on controlling the order in which experiments are executed [Hancox [1980]]. This could be integrated with automatic data collection when it is complete.

End of Chapter

CHAPTER 3: DATA REDUCTION

When working toward the solution of a problem, it always helps if you know the answer - Rule of Accuracy

3.1 Why reduce the Data

Normally there will be four inputs to the ADC. Let the sampling period be one millisecond. The sampling rate per channel will be 250 Hz [1/4E-3]. If the data is recorded over a typical 4 second period, there will be approximately 4K [4s*4*250Hz] samples.

Assume that each sample uses two bytes of storage. The number of tests that can be stored on a 20 megabyte disk will be approximately 2.5K [20M/(4K*2)]. If the daily number of tests carried out at the Clinical Investigation Unit [CIU] is between 25 and 50, it would mean that the 20 megabyte disk could, at most, hold the results of about 50 days of testing!

The above is based on the assumption that there is nothing else on the disk. Typically some 30% of the disk will contain user programs; this implies that the figure will be down from 50 to 35 days! Obviously this is not a practical solution.

An alternative approach would be to 'model' the data and to store the parameters of that model. Using this solution, even if the model had 32 parameters [4 bytes per parameter would be typical of most machines], the number of tests that could be stored on a 20 megabyte disk would be approximately 40K [20M/(32*4*4)]. At the current rate, this implies that the 20 megabyte disk could hold the results for about 2 years of tests [assuming the 30% reduction in available storage].

3.2 The Model

The model used will be purely 'aesthetic', that is, it may not have any significance as far as the actual data is concerned: it will just reproduce the data as closely as possible. From a non-mathematicians point of view, this is quite a task.

A simple model would be a function which would fit the curve. This is normally defined as some linear combination of a set of basis functions, viz

$$y(x) = c_0.p(0,x)+c_1.p(1,x)+ \ldots +c_n.p(n,x)$$

where c_j is the jth coefficient and p(j,x) is the jth basis function.

Typical examples of basis functions are

$$p(j,x) = x^j$$

$$p(j,x) = \sin(n.j.x) \qquad \text{[Chebyschev Series]}$$

$$p(j,x) = \exp(n.j.x)$$

$$p(j,x) = k_0+k_1.x+k_2.x^2+ \ldots +k_n.x^n$$

For convenience, the term 'curve' on its own refers to the real data. The term 'fitted curve' refers to the model.

3.2.1 What is a Smooth Curve?

When attempting to fit a curve, the question that is normally asked is "What type of fit is required?". This question is often met with perplexity, if not frustration: a common reply is "A Smooth Curve". This logically leads to the follow up question "What do you mean by 'smooth'?". The retort to that question usually involves a lot of verbiage, which falls short of answering the question. A typical curve is shown in Figure 3b, and is sigmoidal in shape.

The truth of the matter is that the term 'smooth' can have different meanings for different people:

- The curve could be purely aesthetic, that is, it looks smooth but has no analytical meaning as far as formulation is concerned.
- The curve could be smooth in that its derivatives are continuous [Spline Fit].
- The curve could be statistically smooth
- The curve could pass through all the points

The input signal will be considered before answering this rather ambiguous question. In any input signal from the 'real world', there normally exists some form of noise. In fact, there are two types of noise: background noise and foreground noise. The background noise is caused by the ADC when it attempts to digitize an analog signal: similar to rationalizing an irrational number. The foreground noise is caused by various physiological factors, for example cardiogenic oscillations.

The current trends are towards eliminating all forms of noise. A smooth curve could therefore be defined by a Spline or Statistical Fit. However, later research may require the foreground noise as well as the general shape of the curve. The definition could then change to an 'all points curve' [APC]. Since the consensus may not regard an APC as a 'smooth' curve, the statistical fit will henceforth be used as the definition of a 'smooth' curve.

3.3 Curve Fitting

3.3.1 Curve Fitting Norms

A statistical fit is normally defined by one of the following:

- Minimization of the sum of the absolute errors
- Minimization of the sum of the squares of the errors [Least Squares]
- Minimization of the maximum absolute error [Minimax]

The term 'error' is the difference between the actual curve and the fitted curve at the same abcissa. Of the above, the first method is seldom used. Of the remaining techniques, the minimax norm is often preferred but it can go wrong if there is a lot of noise in the data. This problem arises when an attempt is made to fit the noise rather than the actual curve.

Unlike the minimax norm, the least squares norm attempts to reduce the sum of the squares of the errors. It is therefore less dependant upon individual errors. On the whole, the least squares norm is normally used as it is easier to program and will 'behave' when there is a lot of noise.

3.3.2 Basic Curve Fitting Techniques

There are many curve fitting techniques available. Most of these techniques are based on, or hybrids of, the following:

- Polynomial Curve Fit
- Splines
- Parametric Techniques

3.3.2.1 Polynomial Curve Fitting

As the name implies, the method fits a polynomial to a curve. This polynomial is normally a linear formula of some basis function [examples given in Section 3.2].

Most of the polynomial techniques available are based either on the least squares norm or the minimax norm. A common basis function for the least squares norm is $b(n,x) = x^n$, which results in the polynomial

$$p(x) = a_0 + a_1.x + a_2.x^2 + \ldots + a_n.x^n$$

where a_0 to a_n are the solution of a set of simultaneous linear equations formed from the data. However, this method fails for large values of n [greater than 5] or for large amounts of data. This failure is due to a problem inherent in digital

computers: ill-conditioning. Fortunately, Forsythe [1957] derived a solution using orthogonal polynomials as basis functions. These are polynomials which have similar properties [ie orthogonal]. They eliminate the problem of ill conditioning and give a more accurate result when used on a digital computer. The curve fit using orthogonal polynomials given in Appendix 2.1 is a Fortran IV translation of an Algol 60 routine due to MacKinney [1960] with the modifications suggested by MacMillan [1961] and Makinson [1967]].

Minimax techniques are often mentioned in literature but the algorithm is seldom given. The technique which is often quoted is the Chebyschev curve fit. It uses Chebyschev polynomials as basis functions:

$$p(n,x) = \cos(n.z)$$

$$\cos(z) = x \qquad [x \text{ in the range } -1 \text{ to } +1]$$

Using the trigonometric identity

$$\cos((n+1).z) + \cos((n-1).z) = 2.\cos(z).\cos(n.z)$$

the relation

$$p(n+1,x)+p(n-1,x) = 2.x.p(n,x)$$

$$p(n+1,x) = 2.x.p(n,x) - p(n-1,x)$$

may be derived. If the data is transformed in the range -1 to 1, [the range for which Chebyschev polynomials work] the curves can be fitted in a manner similar to that used for orthogonal polynomials but using the minimax norm. The Chebyschev curve fit given in Appendix 2.2 is a Fortran translation of an Algol 60 program due to Boothroyd [1967].

3.3.2.2 Splines

A spline is defined by Rogers and Adams [1976] as

> "a piecewise polynomial of degree K with continuity of derivatives of order K-1 at the common joints between segments"

Most of the spline fitting techniques available use successive pairs of points as segments. This is a fairly good technique for graphical displays and for fitting APCs.

A derivative of the spline technique is the 'Local Axis Technique'. It uses a transformation to bring the co-ordinates of the curve nearer to the origin before fitting the curve.

Another derivative of the spline technique is B-splines. It uses polygons constructed from the points to shape the curve. The curve does not necessarily pass through any of the vertices of the polygon.

3.3.2.3 Parametric Techniques

This technique redefines the co-ordinates of the array in terms of a third variable [or parameter]. For instance, the circle

$$(x-h)^2 + (y-k)^2 = r^2$$

could be redefined as

$$x = r.\cos(z)+h \qquad y = r.\sin(z)+k$$

A parametric technique which is commonly used is Bezier curves due to Bezier [1972] of Renault. It is useful for showing trends but is not very useful as far as data reproduction is concerned.

- Curve Classification
- Curve Segmentation
- Segmentation and Classification
- Axial Transposition
- Mathematical Transformations

3.3.3 Fitting the Pressure Concentration Curve

3.3.3.1 Curve Classification

The method is based on the idea that there are standard PCCs, and that if all curves are scaled to the same size, their shapes may be matched against these PCCs. This idea was abandoned since there was insufficient data with which to form a standard PCC.

3.3.3.2 Curve Segmentation

The PCC, as it stands can be roughly divided into four regions [Figure 3a]. Unfortunately, the operative word here is 'roughly'. The actual position that separates any two consecutive regions is not clearly defined. In the literature it is often defined as the position where the rate of change of the gradient is significant. This definition is unrealistic as it assumes that there is no noise in the curve.

A similar idea was tried by Payne [1970] for a general curve fitting package; the difference being that the curve was just divided into equal sections. They had no physical meaning. The only requirements were that these sections had to fit a function and that the derivatives at the overlapping points had to be the same.

A similar technique was attempted without checking the derivatives. Instead, the mean of the overlapping sections was used reproduction. Using this technique, a 5% error was obtained in a statistical test on the reproduced curve [see Appendix 3.1 and 3.2].

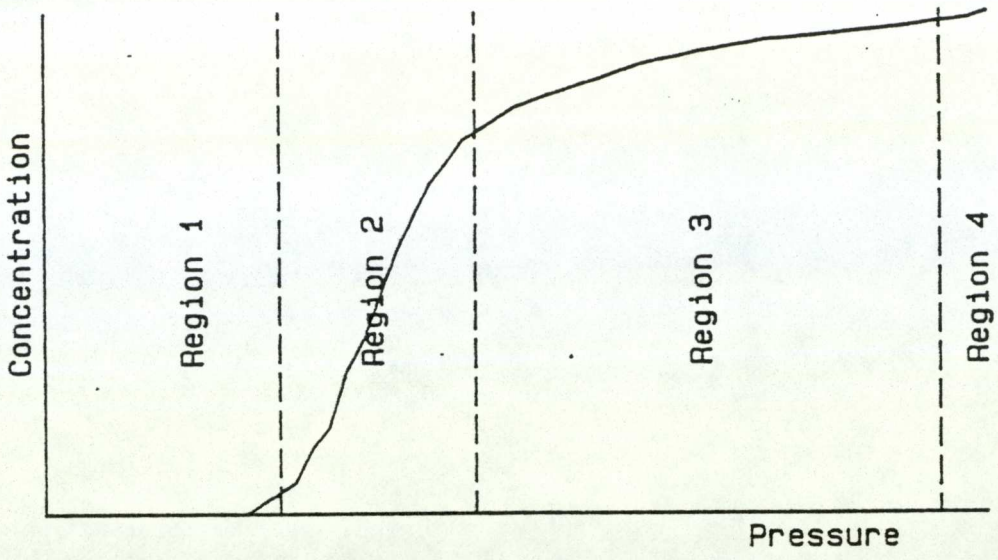


Figure 3a: Regional Separations on the PCC

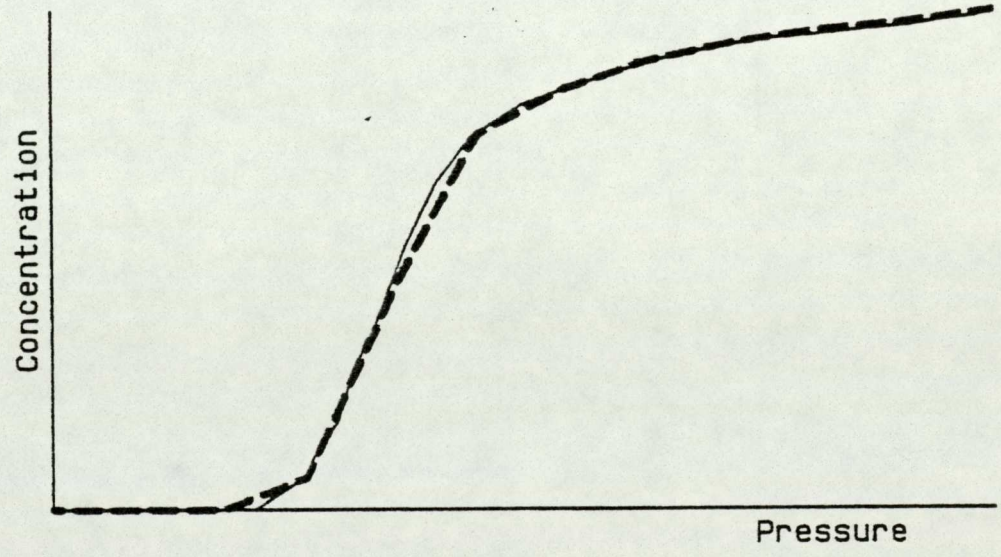


Figure 3b: Approximating a curve to a series of straight lines

3.3.3.3 Curve Segmentation and Classification

Most curves can be approximated to a series of straight lines; this can be easily proven by examining a graph plotter output under a magnifying glass. If the PCC is divided into a series of small blocks, then each block can be approximated to a straight line [see Fig 3b]. Any straight line can be classified given its gradient and its intercept with the vertical axis. If the intercept is always at the origin, then the only identifying feature its gradient.

If the blocks are of fixed size on the horizontal axis, the only two variants are the vertical location of the origin and the gradient of the line. This idea might have worked if the ADC data had been available.

The major problem encountered was whether to have a standard block size or a standard number of blocks. If a standard block size was chosen, then the longer curves would be reproduced more accurately than the short ones. The converse is true if the number of blocks is fixed. Since a compromise could not be reached and data was not available in sufficient detail, this method was abandoned.

3.3.3.4 Axial Transposition

This is a fairly common technique of curve fitting. The curve is fitted to $x=q(y)$ instead of $y=p(x)$. When the PCC is transposed, it looks like a quintic with two points of inflexion. Unfortunately, the fitted function will not always model the curve correctly as can be seen in Figure 3c.

Another problem with axial transposition was that it would be a time consuming task to reproduce the data. The raw data is read as Concentration in terms of Pressure. On reproduction, this will be Pressure in terms of Concentration. A lot of interpolation would have to be done in order to obtain the data as Concentration in terms of Pressure.

3.3.3.5 Mathematical Transformations

This technique was discovered by accident while the author was indulging in one of his pastimes: helping other programmers with their problems. It was found that a quality control formula from Wild [1971], of the form

$$y = (z^n - 1)/(z^m - 1)$$

[where $z=x/(1-x)$ and x is in the range 0.0 to 0.2] approximated to the PCC.

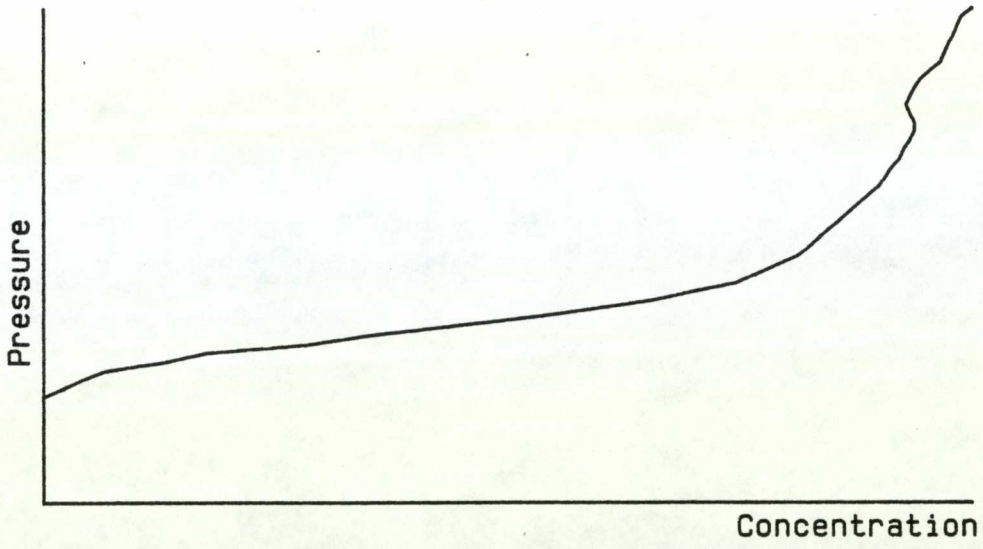


Figure 3c: A curve with transposed axes

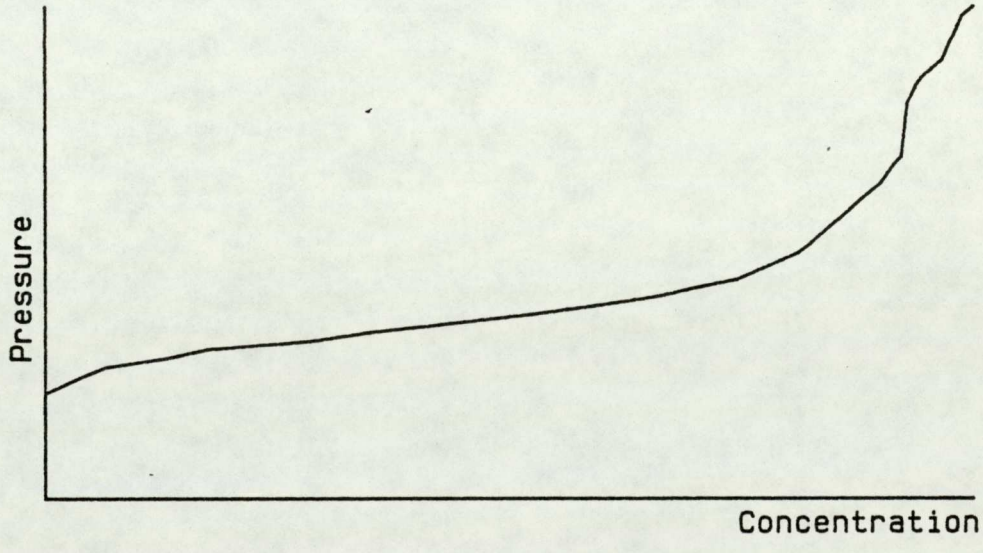


Figure 3d: This curve is a reproduction of the one in Figure 3c when Axial Transposition is used. The 'kink' at the top end is missing.

Further experiments showed that by applying the transformation

$$w = y*x/(2*\max(x)-x)$$

where max(x) is the maximum value of the abcissa, w fitted to a quintic. Since the data was only available in small quantities, it was not possible to test this method fully. However, with about 60 points, it was fairly successful [5% error on a Chi-squared test, see Appendices 3.3 and 3.4].

3.3.4 Fitting an All Points Curve

So far, all the techniques described perform a fit for a smooth curve. Due to the nature of the available data, fitting an APC was not possible. If it were possible to get the data sampled at a fairly high rate, the following steps would have been taken to fit the APC:

- Fit the data to a smooth curve
- Obtain the noise signal by subtracting the smooth curve from the data
- Fit a function to the noise

It is possible the the function will be periodic.

3.4 Testing

3.4.1 The Implementation Computer

The question which arose before any curve fitting was performed was "On which computer should the fitting be done?". It could be performed on either the Information Retrieval Computer [IRC] or the Data Capture Computer [DCC].

If it is done on the DCC, there will be less data to transmit. Curve fitting is a time consuming task so this process could take longer on the DCC than on the IRC. Another problem is that the precision of the calculations on both the computers could be different [the curve has to be reproduced on the IRC]. For the sake of accuracy of reproduction, it was decided that the curve fit be done on the IRC.

3.4.2 Algorithm Testing

The algorithms were tested on a Data General Nova 3/12 running on the Diskette Operating System [DOS] The computer had 64 Kb core of which 20 Kb was used by DOS. The secondary storage was relatively small: only 600 Kb on flexible diskettes. For now, this computer will be called the source computer; the object computer being the final implementation computer which the CIU has yet to obtain. The algorithms were written in Data General's

esoteric version of Fortran IV. These will have to be modified considerably even if they are transferred to a computer which uses Fortran 77.

The curve was reproduced from the polynomials delivered by each algorithm and a Chi-squared test was performed. Initially the basic algoirthms were tested on known curves [eg $y=\sin(x)$, $y=\exp(x)$, $y=x^2-5x+2$ etc]. When these tests gave satisfactory results, they were tested on the PCC. The hybrid algorithms were only tested on the PCC. There was no need to test them on known curves as they were specifically written for the PCC.

From these tests, it was discovered that the reproduction was more accurate if the data was scaled down to the range +2 to -2 rather than 0 to 255. When the process was repeated in double precision arithmetic, a more accurate result was achieved.

Occassionally, during reproduction, large errors appeared in Region 2 [see Figure 3a] of the curve. It was observed that these curves had long 'tails'. Shifting the origin to the position where the curve started rising reduced this error significantly.

3.4.3 Testing on Simulated Data

The algorithms described so far were tested on data read from pen recorder outputs supplied by the CIU. This is equivalent to taking every 20th point from the ADC output. The questions which arose were

- Would the algorithms work if they had been tested on a larger volume of data
- Why was the ADC output not simulated: it could have been compared against the pen recorder outputs for accuracy of reproduction.

A definite answer cannot be given to the first question as no tests were performed. What is known is that the Nova 3/12 would not be able to handle them as there would not have been sufficient memory. This could have been easily solved by shifting the array into secondary storage. However, this would have resulted in a lot of wasted work since secondary storage routines are often unique to manufacturers. It could be a problem programming the equivalent of one manufacturer's routines on other manufacturers' machines. It is hoped that the object computer will have enough workspace to avoid the problem of having to use secondary storage.

As for the second question, the same volume of data could have been simulated using difference tables. The problem was simulating the exact nature of the data. It would not have been feasible to reproduce the pen recorder outputs since the amount of noise generated by the momentum of the pen recorder might not have been equivalent to the noise generated when converting the analog signals to digital signals.

3.5 Further Work

Further work could be done, not on curve fitting but on curve recognition. This would have to be done on the PET. It arises from the fact that an untrained operator would try every single breath test available whereas a trained operator would only try a few as a logical follow up to those attempted previously. The problem would be to cram the trained operators' experience into the logic of a program to aid the untrained operators. This work is currently being attempted at the CIU.

A lot of work could also be done on fitting an APC when the data is available. This is yet an unexpolored area and it would help if some expeditions were carried out in it.

End of Chapter

CHAPTER 4: TRANSFERRING THE DATA

Whenever you set out to do something, something else must be done first - Sixth Corollary to Murphy's Law

4.1 Why transfer the data

In Chapter 2, the data was captured from an ADC by the PET and in Chapter 3, this data was reduced in size by the Nova 3/12. The question is how the data got from the PET to the Nova 3/12. Since the data was read from pen recorder outputs, it did not actually come from the PET, but that was only a simulation of what the PET would have given. The point is, that the data will eventually have to come from the PET. In this chapter, the methods of transferring the data from the PET to another computer will be discussed.

4.2 Methods of Transferring Data

Data can be transferred between computers in various ways:

- Manual Transcription
- Compatible Peripherals
- Direct Linking

Manual transcription involves obtaining a printout from one machine and manually transcribing the data to another. Although this technique is outdated, it is still used by British Telecom for telephone bills. This method is extremely error prone and time consuming, especially when there are approximately 4K numbers on each curve.

The method of common peripherals uses a media which is common to both machines, eg paper tape. The data could be recoded on the paper tape by one machine and read by another: that is, if the formats used by both the machines are the same.

Direct Linking involves the use of one machine as a terminal of the other. The advantage is that one need not worry about compatibility of peripherals. Another advantage is that the software can be easily modified if the host computer is changed; no new hardware need be purchased since all the larger computers nowadays follow the same communications standards.

Of the three, direct linking is the cheapest since it requires no additional stationery and very little manual intervention. It is the least error prone and therefore the most secure of the three methods.

4.3 Communications Ports

In order to use one computer as a terminal of another, something must first be known about the communications facilities available on both these machines.

On most mainframes and minicomputers, the method of communication with the data entry terminals is normally either the EIA RS-232 or the Current Loop Standard.

On the PET, there are two communications ports: the GPIB Port and the Parallel User Port [PUP]. Both these ports can be interfaced with the RS-232 using an interface exchange. The difference is that the GPIB interface exchange can be bought 'off the shelf' whereas the PUP interface exchange will have to be specially built. The GPIB was more attractive as there was no point in re-inventing the wheel. As a result, the GPIB port was selected as the communications port for the PET.

4.3.1 The GPIB and its Protocol

GPIB is the abbreviation for General Purpose Interface Bus. It is an international standard which has been developed so that programmable instruments made by different manufacturers, but all following this standard, can communicate easily with one another.

The GPIB is also known by several other names:

- HPIB [Hewlett Packard Interface Bus]
- ANSI MC 1.1 [American National Standards Institute, Standard MC 1.1]
- IEC Bus [International Electro-technical Commission Standard 625-1]
- IEEE-488-1978 [Institute of Electrical and Electronic Engineers Standard 488-1978 Digital Interface for Programmable Instrumentation]

The interface was originally developed by Hewlett Packard [hence HPIB] and later adopted by the IEEE, ANSI and the IEC. The title GPIB was given by Tektronix. The main reason for calling this bus the GPIB rather than the IEEE-488 bus [which the PET manual uses] is that it is a shorter abbreviation.

The GPIB allows the transfer of digital data among up to 15 devices connected together either by a star interconnect or party line bus [see Figures 4a and 4b]. The total length of the interconnecting cables must not be more than 20 metres. The data is transferred in parallel format [bit parallel, byte serial] in an asynchronous fashion. The speed of each transaction on the bus is set by the slowest device participating in that transaction. The maximum data rate allowed by the standard on any signal line is one million bits per second.

Three types of devices can exist on the GPIB: talkers [transmitters], listeners [receivers] and controllers. In a PET system, the PET is the only controller on the bus.

The GPIB works on a 'three wire handshake' [protocol] using the following control lines:

- NRFD Not Ready For Data
- NDAC Not Data Accepted
- DAV Data Valid

The protocol works as follows:

- The transmitter waits until all the listeners it wishes to talk to have released NRFD.
- The talker then places the data onto the data bus and activates DAV. This tells each listener to capture the data byte presently on the bus.
- NDAC is held low by each listener until it captures the data byte. When all listeners have read the data, NDAC goes inactive telling the talker to take the byte off the data bus.

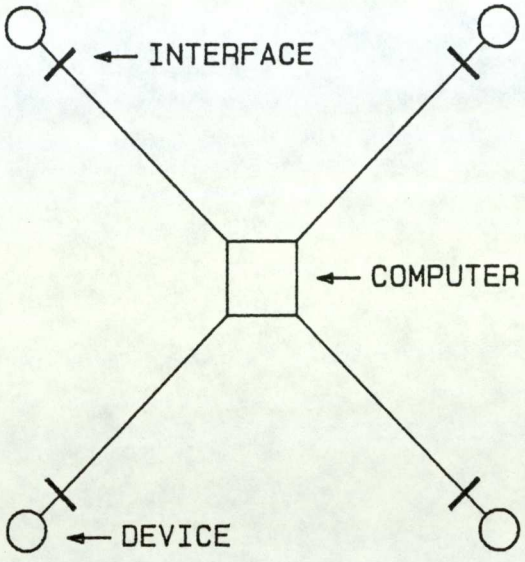


Figure 4a: Star Configuration

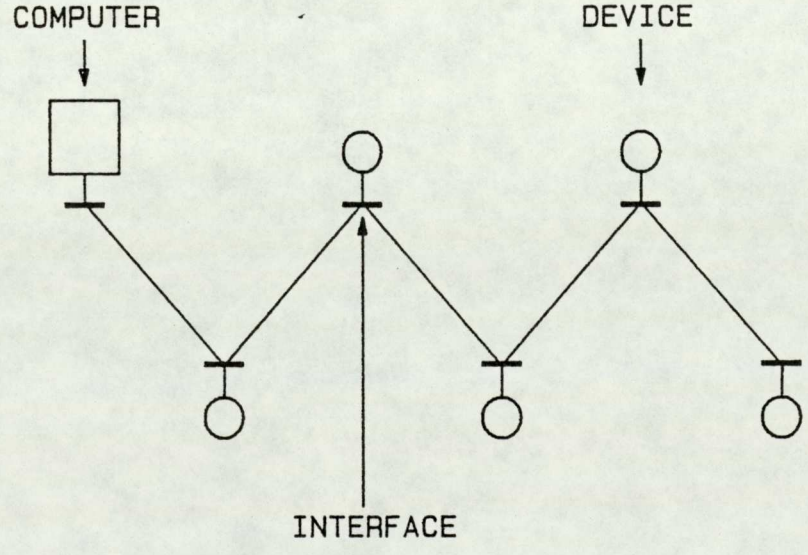


Figure 4b: Daisy Chain Configuration. The GPIB can be used in a star, daisy chain or permutations of both configurations with up to 15 devices.

4.3.2 The RS-232

The full name of the RS-232 is the Electronic Industries Association [EIA] Standard RS-232: Interface between Data Terminal Equipment [DTE] and Data Communications Equipment [DCE]. It is also known as the International Telephone and Telegraph Consultative Committee [CCITT] Standard V.24.

It is a convention for data transmission over short distances. It was originally designed to interface terminals to telephone modems, but is now widely used for the interconnection of computers and peripherals of all sorts.

The RS-232 uses a 25 pin connector, with the plug tied to the DTE and the socket to the DCE. The transfer of data is accomplished by sending the data in serial form [one bit at a time] over a single wire. There are two general transmission techniques: synchronous and asynchronous.

Synchronous data transmission requires a clock to mark the start of each data bit interval. There are also special bit synchronization patterns which may be used to allow the receiver to locate the first bit of the message. The disadvantage of synchronous transmission is that the data must be contiguous.

With asynchronous transmission, a clock signal is not transmitted with the data and the chararacters need not be contiguous. This is made possible by adding synchronizing start and stop elements to each character. Asynchronous transmission is usually used for DTE.

4.3.3 The Current Loop

The Current Loop is also known as the CCITT V.28 Standard. It is not as common as the RS-232. The difference between the Current Loop and the RS-232 is in its operation. Both the standards use the same 25-pin connector. The advantage of using either the Current Loop or the RS-232 is that no handshake is required.

4.3.4 The Interface Exchange

Two interface exchanges were tested on the system. One was the IEEE-488 to RS-232-C Bidirectional Serial Interface by Small Systems Engineering [SSE] and the other, the TNW-3000 by TNW Corporation. They are both similar except that the TNW interface will accept two input/output channels while the SSE interface will only accept one input and one output channel. Both these interface exchanges can be configured for Current Loop operation if required. The TNW-3000 is part of another project which had been postponed due to lack of staff.

4.4 Transmitting the Data

There are several items to consider before the data can be transmitted; these are given as follows:

- The message length
- Transmission Codes
- The message format
- Separators and Teriminators
- Error checking
- Job Control Language Protocol

4.4.1 The Message Length

Normally, when an operator wishes to send a message to a host computer [HC] via a VDU, the operator will type in a line followed by a line terminator [eg return]. In most systems, there is a maxmimum length for this message. For instance, it is 2000 on the ICL, 132 on the Nova 3 etc. The problem is to select a message length which will be acceptable to most machines.

These messages have to be read by the HC using either its 'input' command or a program [some computers do not have an 'input' command]. If it is a program, the same problem arises: different implementations of different programming languages will accept different message lengths.

The easiest way out of this is to use the smallest terminal width of a common DTE. This would be the Teletype 33 with a terminal width of 72 characters.

4.4.2 The Message Format

The data, as it is held in the PET diskettes, is in byte format. Since the data bus [GPIB] is only 8 bits wide, the data will have to be transferred one byte at a time. The problem is to establish a format so that the PET 'knows' what to transmit and the HC 'knows' what is coming in. The captured data could be converted into floating point before transmission but that would result in a loss of precision. As a result, two types of data have to be considered:

- Integer Data
- Floating Point Data

4.4.2.1 Integer Data

A solution would be to transmit the data in the format used internally by the HC but there are problems:

- Some of the bit patterns may have a predefined meaning to the Operating System [OS] used by the HC. For example, 007 could be used to sound the buzzer.
- The OS may require the parity bit for error checking; in which case only 7 bits of data may be transmitted.
- Some computers use sequences of characters to terminate files. If the data transmitted happens to contain this terminating sequence, the input could be terminated prematurely.

Consider the program which will have to read this data. The easiest format to read would be decimal. Unfortunately, it is relatively difficult to convert a number to decimal if it is stored in binary. It would be easier to convert the number to base 4, octal [base 8] or hexadecimal [base 16]. On the receiving end, hexadecimal would be extremely awkward to convert but octal and base 4 do not present any problems.

The problem now is whether the data should be sent in octal, base 4 or binary. Since octal numbers can be represented in the least number of bytes, it is the most feasible of the three. The digits will have to be translated into the required transmission code before they are transmitted.

A 12-bit number can be represented in 4 octal digits. This would mean that up to 18 [72/4] numbers can be packed into a line. Unfortunately, some programming languages required separators. To be safe, two spaces will be used. This brings the width of the number up to 6 characters and the number of numbers per line down to 12 [72/6].

4.4.2.2 Floating Point Numbers

Unlike the integer data, the floating point data is not so easy to transmit. It would not be a good idea to transmit it in octal as there are a variety of formats for floating point octal numbers. An easy way out would be to use the STR$ function in PET Basic and convert the number to decimal before tranmmission. It was mentioned eariler that this might have resulted in a loss of precision but that is only in the calculations.

The data input by the operators would not normally contain more than five significant digits [SD]. The precision of the PET is 10 SD whilst that on most 16-bit minicompters is 7 SD [14 SD if double precision is used]. Since the precision of both the machines involved is greater than that of the data, there will not be any significant loss in precision.

The only problem with the data is that it might be difficult to read [from a program on the HC]. Some languages have an unformatted 'read' instruction. Others like FORTRAN IV have a 'G format' which will take in both fixed and floating point numbers in a formatted form. Problems would arise if languages like COBOL and RPG were used since a parser would have to be written to evaluate the acutal value of the number.

4.4.3 Transmission Codes

When two computers pass streams of bits between each other, some coding convention will be required so that the data received may be interpreted. This can be done by breaking the stream of bits into packages [commonly known as bytes or characters].

These packages will always contain a fixed number of bits and some bit patterns will have a predefined meaning [eg 100 could mean End of Transmission]. The problem is that the predefined meanings used by different computers could be different.

As it stands, the PET uses a code unique to its range of machines. Most other machines use the International Standards Organization [ISO] Alphabet No. 5. This standard is often called the American Standard Code for Information Interchange [ASCII]. It was introduced by the ISO and ANSI in an attempt to forstall an impending proliferation of codes and has been fairly successful. There are two solutions to this problem of incompatibility:

- Use the HC to translate the code transmitted by the PET
- Use the PET to translate the code before transmission

Of the two, the latter is easier since the PET character set is larger than ASCII and the PET, as an independent computer can work at a faster rate than the HC. To conclude, the PET will be used to convert the data to ASCII before transmission.

4.4.4 Separators and Terminators

Whenever data is transmitted, it is possible that more than one set that is one complete curve will be sent. A separator will be required to separate one set from the other. Although there is a standard ASCII character for this, it may have a predefined meaning to the OS. An octal number cannot be used since 0000 through 7777 are all used. However, there is no rule that says that the separator has to be an octal number; it could be 'END' or some other decimal number which cannot be converted into octal

[eg 9999]. Since the data on the HC can be read as decimal numbers, invalid octal numbers could be used as separators.

The 'end of data' marker is not really necessary as most computer languages have some way of detecting 'end of file'. However, if there is no such facility, another invalid octal number [say 8888] could be used.

4.4.5 Error checking

Most systems have a method of checking their data. Some use parity bits while others use special techniques such as checksums and cyclic redundancy checks. However, these special techniques are seldom performed during manual data entry.

Since the hardware already caters for parity and the more sophisticated forms of testing will not be used, there is no need to restructure the data for error checking. The error rate will be similar to that of an operator at a remote terminal and this is relatively low.

4.4.6 The Job Control Language Protocol

The protocol for most JCLs is vaguely similar. It consists of a command followed by a command terminator [normally return]. The format and contents of the command may vary from system to system. They could be either left as part of the interfacing software or provided as a parameter to the system.

4.5 Programming the system

The system can be programmed in one of three ways depending on what is available on the HC:

- The PET could emulate an operator typing in an 'input' command and manually entering the data into a file. The data can then be translated into the format required by the HC in a separate program.
- A program could be written on the HC to put the data into a file. This is similar to the above method except that a program is used insead of the 'input' command.
- A program could be written on the HC to read the data from the PET and to convert it before entering it into the file.

The first method is useful if an 'input' command exists in the job control language of the HC. The latter methods depend on the speed of the processor. The method which uses two programs would be better for a slower processor whilst the method which uses concurrent programs would be better for a faster processor. All three methods have been implemented.

4.5.1 Timing

In a test run written in Basic on the PET, an attempt was made to connect the PET as a terminal of the Nova 3. The problem with using Basic was its speed. The program was too slow to accept any messages which had been sent to it and too quick in sending its own messages! This meant that the program had to be written in assembler.

There was a choice of either writing the whole GPIB protocol as described in Section 4.3.1 or using the PET's input/output routines. The former would give better control over the timing whilst the latter would be easier to write. Fortunately, a package existed for such a purpose. It was part of the postponed TNW-3000 project and did not have to be bought separately.

4.5.2 Commlink

The package which was used to handle the protocols was Commlink, supplied by Taylor-Wilson Systems [1980]. It provided a range of commands which configured the PET as a terminal. The problem with Commlink was that it was supplied on a diskette which had been protected by Commodore against piracy. As a result, the research was hindered by the availability of this diskette.

This problem was quite easily solved by loading Commlink into core and saving the core image. Since Commlink used an extended instruction set, it was obvious that the PET's scanner [the CHRGET routine in Commodore's terminology] had been altered. This was also copied and a program was written to load both these core images into their appropriate places. The loader is given in Appendix 4. So much for Commodore's software protection!

4.6 Conclusion

In conclusion, this section of the project has been a very worthwhile excercise in communications. It shows the importance of communications standards in industry and the versatility of general purpose tools.

End of Chapter

CHAPTER 5: STORAGE AND RETRIEVAL OF DATA

Any given program will expand to fill all available memory - Fifth Law of Computer Programming

5.1 Introduction

In this chapter, when the word 'information' is used, it means 'the data collected from the ADC in reproduced form and some facts about its source [eg age, sex, diseases etc]. The main concern of this chapter is the collection of information in the computer's memory and its subsequent retrieval.

5.1.1 The Information Retrieval Systems Available

There were only two information retrieval systems available with no additional cost to the project. These were Rapport by Logica Ltd. and MUMPS (acronym for Massachusetts General Hospital Utility Multi-Programming System).

5.1.1.1 Rapport

Rapport [Logica [1979a,1979b,1979c]] is a package which can be interfaced with Cobol, Fortran and Coral, thus providing some flexibility as far as the machine was concerned. However, its query facilities were limited in that the only boolean operation allowed was AND. This is not a problem as any boolean expression can be easily manipulated [using De Morgan's Laws [see Section 5.4.2.2]] to give the same meaning.

The main problem is that the data would have to be restructured into Codd's Third Normal Form which is rather awkward as it would not handle repetitive types. All the retrieval permutations had to be known before the system was set up. This would have been adequate for the operators who carried out the breath tests but it would have been a drawback for the researchers.

5.1.1.2 MUMPS

Since MUMPS [Digital Equipment Corporation [1976a,1976b]] is now supported by ANSI, it is available on quite a few machines. It is very much better than Rapport in that it will do ANDs and ORs. The problem with MUMPS is that it is a language in its own right and therefore cannot be interfaced with other languages. As a result, the data transferred into a file in Chapter 4 may have to be manipulated so as to get MUMPS to accept it.

MUMPS is an interpretive language and is geared for efficient retrievals. It would not be efficient to use it for heavy arithmetic such as curve reproduction/fitting.

Another problem was that the version of MUMPS available could only accept a record size of 132 characters. This was inadequate as the parameters for the curve alone would use that amount of space. It was only available on a bureaux basis which meant that whoever used it had to follow the times of the bureaux. It would be expensive to use such a system for research work.

5.1.1.3 Other Information Retrieval Systems

The other information retrieval systems available are unique to manufacturers. Some of them can be interfaced with programming languages. The problem is that the idiosyncracies of these systems could hinder the development and portability of the project. It would therefore be better if a system was designed for storage and retrieval of the data.

5.2 The Descriptive Tools

5.2.1 Terminology

Before going on, some of the terminology used will be explained. Each subject [patient] is associated with a set of data. This set of data is called a 'record'. Each record has either a special set of items or a special item which can be used as a unique identification. These special items are known as the primary keys.

A collection of records is a 'table' or a 'file'. The term table is used to indicate a small file while a file is used to indicate a large table. Normally files are stored externally while tables are stored in core. A large file or group of files is called a 'databank' or a 'datapool'. The word 'database' is sometimes used but it will be avoided as it has a range of meanings. The information referred to in Section 5.1 will be stored in a data bank.

Some 'tree' terminology will be used later in this chapter. This terminology is explained in Figure 5a.

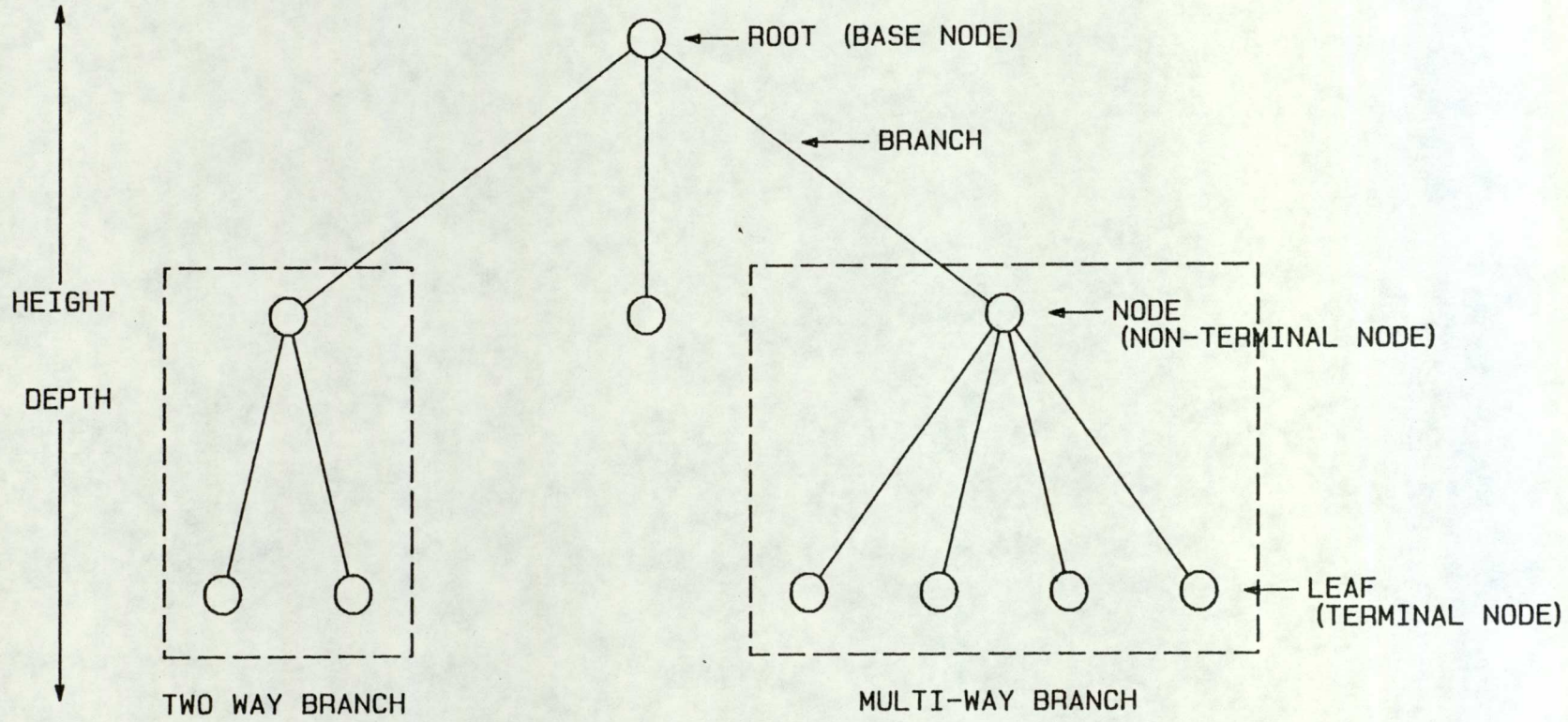


Figure 5a: Tree Terminology

5.2.2 Data Structures in Algol 68

Algol 68 was chosen for defining data structures because it is one of the few computer languages in which a data structure is not called a 'record'. If Pascal, Ada or Edison had been used, the appearance of the keyword 'record' would have been quite confusing. The use of level numbers as in Cobol and PL/1 just hinders description.

Algol 68 has been used extensively by Pagan [1976,1979,1981] for describing the semantics of programming languages. In this section, some of the tokens used in the later sections will be described. A data structure is defined by

```
MODE structure name = STRUCT(structural descriptor)
```

The token STRUCT may be left out if there is only one item in the structural description. The structural descriptor is a set of structure names. Arrays in Algol 68 are described by brackets [], thus [1:20]INT x is an array of 20 integers, the array name being x. The word REF means that the entity is used as a pointer. This is illustrated in Figure 5b.

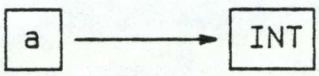


Figure 5b: Diagrammatic representation of REF INT a.

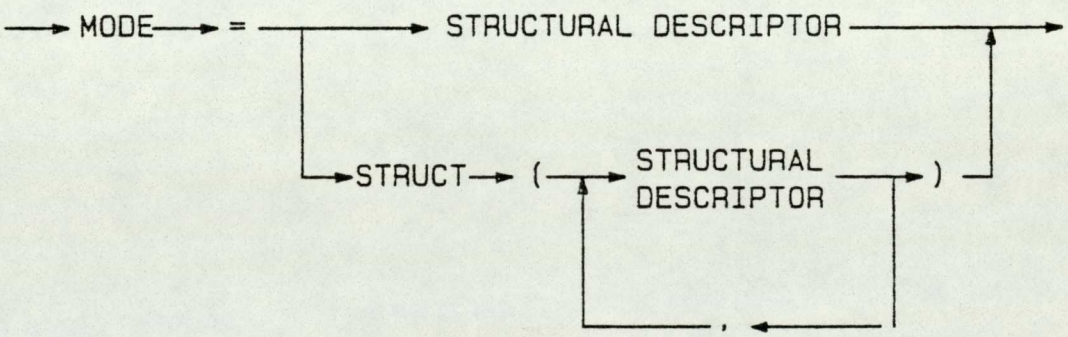


Figure 5c: Syntax Diagram of a MODE declaration in Algol 68.

Sometimes a mode can be defined as one of a series of types. This is symbolized by the word UNION. For instance, if an entity can be either a STRING or an INTeger, then it may be defined as

```
MODE ENTITY = UNION(STRING,INT)
```

5.2.3 Syntax Diagrams

In Algol 68, the order in which the structural descriptors appear is not important. Since the order is important in the syntax of most languages, Algol 68 cannot be used for syntactic description. Instead, a system of describing syntax using diagrams due to Wirth [1976] will be used. The syntax is described in the direction of the arrows [see Figure 5c].

The syntax could have been described in Backus Naur Form [De Morgan et al [1976]] but this would have introduced problems of ambiguity of grammars et cetra. Such problems are not in the best interests of this report.

5.3 The Data Bank

5.3.1 Maintenance of the Data Bank

Most data banks can be maintained with the aid of the following operations:

- Insert
- Update
- Delete

These operations all require yet another operation: retrieve. This is the most important operation as it is used not only in the above but also as an operation in its own right.

5.3.1.1 The Insert Operation

This operation involves the addition of new data in order to keep the file up to date. It is normally implemented as a search for a space in which the record may be inserted followed by the actual insertion of the record. If the record has to be put in a specific place to ease the searching, other records may have to be shifted to accomodate this insertion.

5.3.1.2 The Update Operation

This operation is used when data within an existing record has to be modified. The record stored in the file will contain some information from the previous record and possibly some new information.

5.3.1.3 The Delete Operation

When data is no longer required in a file, it can be deleted. Although this operation is not normally required for a cumulative data bank, it is useful for removing bogus records which are often inserted during demonstration runs.

5.3.2 The Retrieve Operation

In order to use any data stored in the data bank, the record must first be read by the processor. To read a record efficiently, its contents must be located quickly. Records may be retrieved in one of two ways:

- Using the primary key only
- Using permutations of secondary keys

A secondary key is an item which is not part of the primary key. It may be used for retrieval, but unlike the primary key, it need not be unique. When a retrieval is performed on the primary key, only one record will be expected since the primary key is unique. A retrieval based on secondary keys could produce more than one record. If at least one record was found after the search, it would be 'successful'; conversely, if no records were found, it would be 'unsuccessful'.

5.3.3 Searching on the Primary Key

There are several ways of searching on primary keys. Some are better for retrievals while others are better for insertions and deletions. Some techniques are the converse. Fortunately, there are some techniques which strike a compromise between file maintenance and record retrieval.

The idea behind these techniques is to locate the required record with the minimum number of disk accesses [since the time required to access a disk block is often very much greater than that required to manipulate it]. The methods of file searching that will be discussed are

- Indexed Sequential Access Method [ISAM]
- Key Transformations
- B-trees

5.3.3.1 Indexed Sequential Access Method

The ISAM of data organization allows the data to be processed either directly or sequentially [unlike sequential organization which only allows sequential access]. It uses a file call called an Indexed Sequential File [ISF]. The ISF has three areas:

- An index
- A sequential file
- An overflow area

The index is a collection of sorted primary keys and pointers. Each pointer corresponds to a primary key and points to the remainder of that record stored in the sequential file. The index is often organized into ranges [for example, 1000 to 2000, HOW to HUG] called sub-indices. These indices help to speed up the search.

The sequential file contains the secondary keys and other non key information. The overflow area is used when the space in any sub-index is exhausted [the ICL solution] or when the whole index gets filled up [the IBM solution]. It is basically a temporary fixture to avoid file reorganization. However, when the overflow area gets filled up, the file has to be reorganized.

5.3.3.2 Key Transformations

This is a relatively old idea which first appeared in the early 1950's. In most search techniques, lists, tables or trees are scanned for matching primary keys. When the primary key is found, the next item that will be accessed is a pointer to the remainder of the record. The idea behind key transformations or hashing is to use a transformation function to compute the value of that pointer, given its primary key.

Theoretically, only one memory access would be required to obtain the record. Unfortunately, it is not easy to find such a function. Knuth's [1968] mathematical treatise shows that the chances of finding a function which will uniquely map 31 keys to a table of 41 records is one in a million! As a result, the probability of a function computing the same value for different keys is very high. Such duplication of values is technically called a 'collision'.

Fortunately, there are several techniques for handling collisions: linear probing, quadratic probing, chained lists etc. There is no 'best' general method of handling a collision. The use of a key transformation has several disadvantages:

- It is extremely data dependant.
- Its performance is affected by the size of the hash table; a larger table gives better performance.
- The size of the table needs to remain fixed: this could delay the introduction of new hardware as the size of the table may have to be altered to take advantage of this addition. Normally the hash table has to be reorganized.
- The records cannot be accessed sequentially unless the transformations specially cater for this.
- Some collision handling techniques will not work if keys are deleted.

5.3.3.3 B-trees

A B-tree is basically a balanced multi-way tree. It was introduced by Bayer and McCreight [then of Boeing Corporation] in 1970. With a B-tree it is possible to search and to update a large file with 'guaranteed' efficiency: even in the worst case.

Knuth's description of a B-tree of order n is given below:

- Every node can have up to n+1 branches
- Every node, except the root and leaves, has more than n/2 branches
- A root has at least 2 branches [unless it is a leaf]
- All leaves appear on the same level and carry no information
- A non-leaf node with k branches will contain k-1 keys

A more detailed description of the B-tree is given by Comer [1979]. The main difference between Comer's description and Knuth's description is that Comer's B-tree of order n has 2n+1 branches. A diagram of a B-tree is given in Figure 5d.

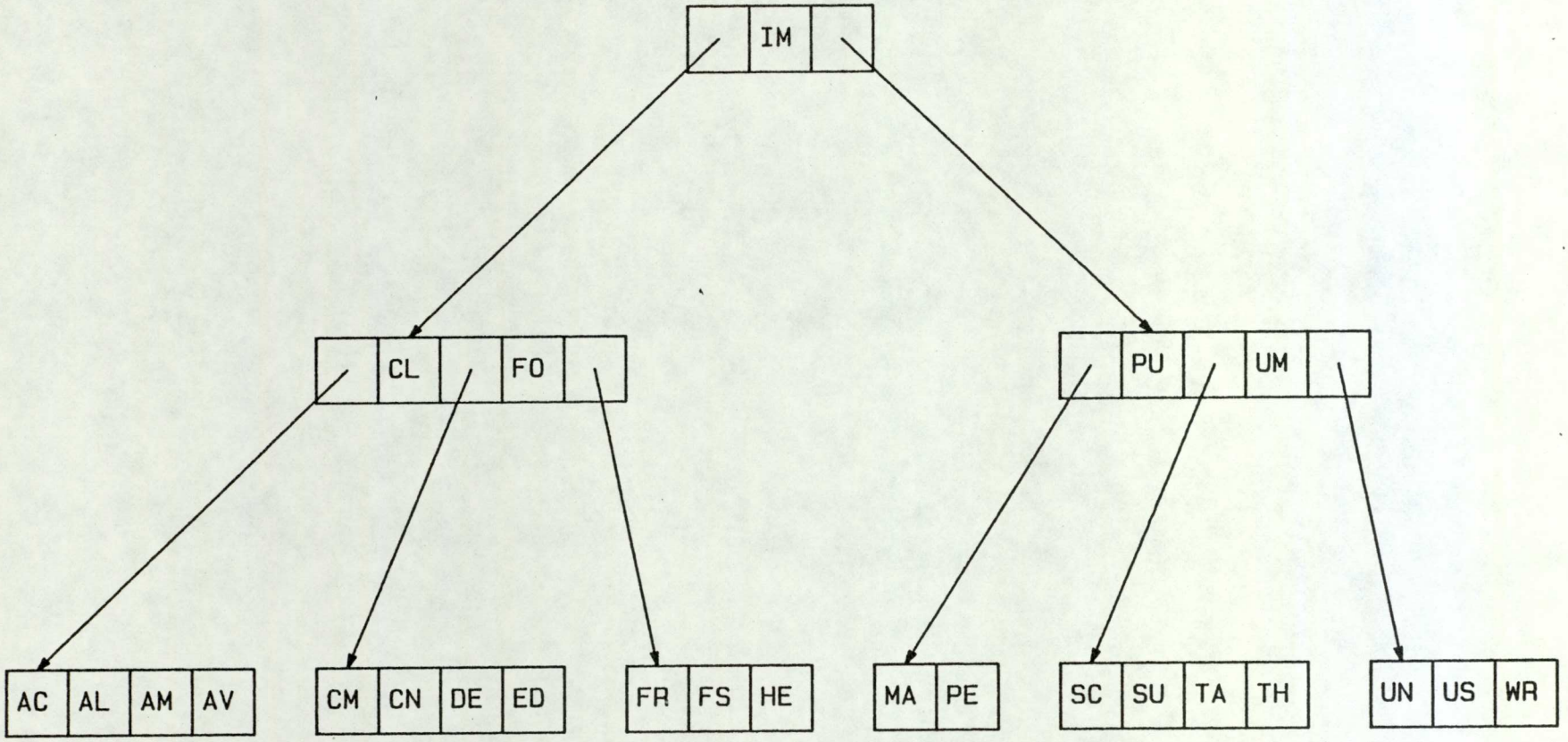


Figure 5d: A B-tree.

Using a B-tree has some advantages:

- Records can be accessed in relatively few disk accesses. Say, the contents of a node can be stored in one block. This implies that only one access will be required for each node. Therefore only four disk accesses will be required to pick out one record from 11110 in a B-tree of order 10!
- Each node has its keys in sorted order. This implies that the records may be accessed sequentially.
- The tree is always 'balanced', that is, the depth of each subtree is always the same. This guarantees a minimal time to access each record.
- It automatically reorganizes itself.

Incidentally, the B in B-tree has no definite meaning. Many references refer to it as a 'Bayer-tree' although it could stand for balanced, bushy, broad or Boeing! An AVL tree (balanced binary tree) is a B-tree of order 1 whilst a 2-3 tree is a B-tree of order 2.

5.3.3.4 Selection of retrieval technique

When searching for a record based on a primary key, one of the following possibilities can arise:

- the primary key is known
- part to the primary key is known but the operator is uncertain about the rest of it
- the primary key is known but the operator is too lazy to type all of it in!

If the primary key is known then any technique may be used. The latter two are basically the same but they imply that a sequential search is required so hash functions may not be used.

Since ISFs need occassional manual reorganization while B-trees reorganize themselves automatically, it is more convenient to use B-trees. To cater for half entered primary keys, wild characters and coded strings [eg cm for centimetres] could be introduced into the system.

5.3.4 Retrievals on Secondary Keys

When retrieving on secondary keys, a search is made for all the records which have certain attributes. The specification of the records required is known as a query. Queries are usually restricted to the following types:

- A simple query which gives a specific value of a specific attribute; eg sex eq male.
- A threshold query which gives a limit to the values of a specific attribute; eg age gt 50
- A Boolean query which consists of the previous query types combined with Boolean operations;
  eg sex eq female and age lt 20

There are several methods of performing queries efficiently (eg combinatorial hashing, clustering etc). Unfortunately, these methods will only work if the distribution characteristics of the attributes are known. There is also a possibility that some records satisfying the query will be left out. Such a fault could result in a multitude of bogus theories, especially if this data bank is used as a data source. There are, however, two simple and straightforward solutions to this problem:

- Record Search
- Attribute Search

Both these methods use a basic search technique known as the sequential search. Although they are not as efficient as clustering and the like, distribution characteristics need not be known.

5.3.4.1 Record Search

In a record search, the file is organized into records viz

Record 1	Brandy	Napolean	40%
Record 2	Brandy	Rouget	40%
Record 3	Whisky	Glenfiddich	40%
Record 4	Liquor	Benedictine	38%
Record 5	Whisky	Bells	41%

When a query is processed, each full record is read in turn and matched against it, the operative word here being 'full'. A lot of time is often wasted in reading the whole record, most of which is not required for the query.

5.3.4.2 Attribute Search

This is the technique used in the programming language APL. Instead of keeping the keys or attribues of a record together, each attribute is kept separately viz

Attribute 1: Brandy, Brandy, Whisky, Liquor

Attribute 2: Napolean, Rouget, Glenfiddich, Benedictine

Attribute 3: 40%, 40%, 40%, 38%

When a query is made, only the attributes which are required are accessed. This speeds up the search process considerably. Another advantage of such a structure is that new attributes can be added to the system without any complications. The main disadvantage of such a system is that several blocks have to be read when a 'full' record is required.

5.3.4.3 Choice of Search Technique

A decision has to be made as to whether a quick search or the speed at which a 'full' record is displayed is more important. Most queries will only require parts of records: not the whole record. Normally, the time spent comparing records which do not satisfy the query is very much greater that the time taken to access several blocks of data.

However, if the record has several attributes, then the time taken to access several blocks of data can still be considerable. To get the best of both worlds, a hybrid technique can be used. Instead of keeping each attribute separately, some attributes could be grouped together. This would increase the search time marginally but it would decrease the access time considerably.

5.4 Interacting with the Program

5.4.1 Methods of Interaction

There are two standard methods of interacting with a program:

- Using a Menu System [MS]
- Using a Command or Query Language [QL]

A QL provides the basic building blocks with which the user can build up whatever is required. This approach is normally preferred but, as with all other languages, it is initially more difficult for the user.

A MS is one that provides a list of options. The user can then select as many options as required; very much like selecting dishes from a menu in a restaurant.

Most MSs attempt to provide the user with every conceivable option. Unfortunately, this implies that an extremely large menu could result and that it might not provide options which are required. An alternative MS could provide the user with QL facilities but this could be extremely clumsy to use.

The choice of interacting method lies in the application: not in the merits of the method. If the user is not likely to require more than two operations at any given time, a MS should be used. On the other hand, if more than two operations are required, a QL should be used.

5.4.2 Choice of Interacting Method

There are two groups of operations that can be performed on the data bank

- Maintenance
- Retrievals

As a rule, the maintenance operations should only be performed with the use of the primary key only. This will avoid the problem of duplicate primary keys. Since these are single operations, a MS should be used for mainentance.

In retrievals, the data bank will not be altered in any way so either a QL or an MS may be used. This is based on the assumption that most queries will be small. There is a possibility of a large query but it is just a bad excuse to introduce a QL into the system.

Imagine a researcher sitting in front of a VDU. Unlike the operators, this researcher does not use the system very often and makes logic errors like typing IOR instead of XOR. If the search does not give the researcher the records which are required, the whole query will have to be re-entered.

With a MS, the researcher will have to go through screens of menus all over again in the order in which the query should have been evaluated. Such a system could get rather 'unfriendly'. With a QL, the researcher is given a notation with which to record the query. This aids remembering the query. The computer can help by displaying the original query in the QL. Using a QL would therefore solve some problems for the researcher. In conclusion, a QL should be used for retrievals.

5.4.2.1 Design of the Query Language

One of the reasons for using a QL is flexibility. It should allow the user to configure any required query. Its notation is normally based on one or more of the following:

- Infix Notation [IN] eg 5 * 4
- Prefix/Polish Notation [PN] eg * 5 4
- Postfix/Reverse Polish Notation [RPN] eg 5 4 *

The differences lie in the location of the operator [eg *] and the operands [eg 5]. Since Infix Notation is always taught in schools, it is the preferred notation. Polish Notation [so called after the country of origin of J. Lukasiewicz, the originator] is normally used with unary operators and functions [eg tan, cos, log etc]. Its other common use is in artificial intelligence languages.

RPN should be well known to computer scientists and owners of Hewlett-Packard calculators. It is also used in computer languages like POP2 and FORTH.

Both PN and RPN are less error prone but they are difficult to introduce to first time users. IN is easy to use for simple expressions. However, one tends to get lost in parentheses when it comes to complex expressions, for example,

((((age gt 5) and (age lt 10)) or (name eq bloggs) xor
((age eq 20) and (name eq other))) [xor means exclusive or]

Instead of using parentheses, level numbers could be used [an idea from COBOL]. For legibility, each expression will be written on a separate line and indented.

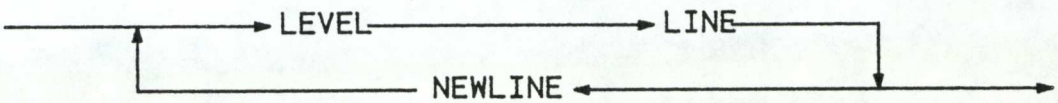
```
          4  age gt 5
       3 and
          4  age lt 10
    2 ior
       3 name eq bloggs
1 xor
       3 age eq 20
    2 and
       3 name eq other
```

This, however, is not so easy to read; moreover, the level of nesting must be determined before the first item is typed. The same problem arises with RPN, but with PN, the level can be easily evaluated. The simple expressions, that is, those with one operator only, will be left in IN.

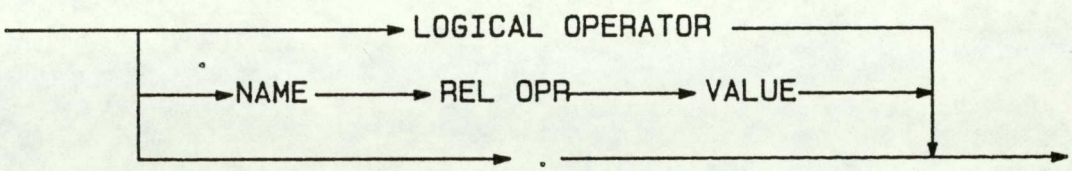
```
1 xor
    2 ior
        3 and
            4 age gt 5
            4 age lt 10
        3 name eq bloggs
    2 and
        3 age eq 20
        3 name eq other
```

The syntax of the query language is described in Figure 5e.

QUERY

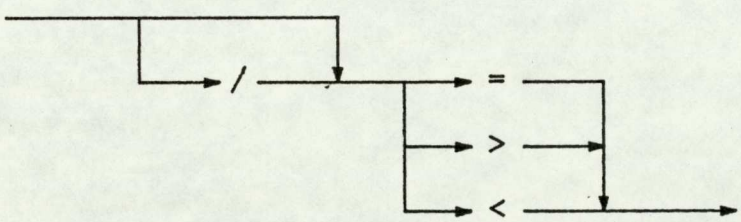


LINE



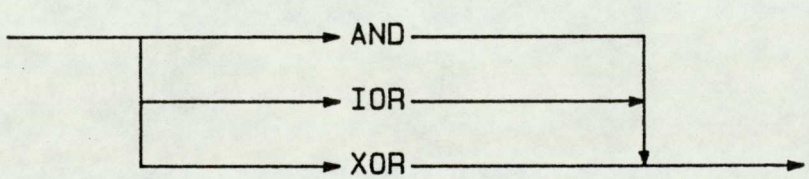
. denotes 'End of Query'

REL OPR



/ means 'NOT'

LOGICAL OPERATOR



IOR means Inclusive OR
XOR means Exclusive OR

Figure 5e: Syntax diagrams for the Query Language.

5.4.2.2 De Morgan's Laws

These are laws for simplifying boolean operations. They are especially for rephrasing expressions using operators which are present only in the QL. There are two basic laws:

- (not a) and (not b) = not(a or b)
- (not a) or (not b) = not(a and b)

Take for instance, an attribute a in the range 10..15,

q = not((a ge 10) and (a le 15))

using ng to represent not greater and nl to represent not less,

q = not((a nl 10) and (a ng 15))
= not(a nl 10) or not(a ng 15)
= (not not (a lt 10)) or (not not (a gt 15))
= (a lt 10) or (a gt 15)

which can be translated into the QL as

```
1 ior
  2 a lt 10
  2 a gt 15
```

5.4.2.3 Processing the Query

Processing an expression in PN requires the maintenance of two stacks; one to keep track of the operands and another to keep track of the operators. With RPN, however, only only one stack is required. RPN also happens to be PN in reverse which is quite convenient for parsing.

The IN used in simple queries can be easily converted to RPN when analyzing the syntax of the input but it not necessary as binary operators can be processed fairly easily. Having fixed the method of processing, an algorithm can be formulated. This is given in Appendix 6.

5.4.3 The Editor

As with any practical system, the user is seldom satisfied with the first attempt. Perhaps a line was left out or an IOR should have been used instead of an AND. As always there is more than one solution:

- Let the user re-type the query
- Provide a facility for the user to edit the query

The first solution is easier for the implementor: no extra coding is required. However, the psychological unfriendliness of the system grows with the length of time taken to formulate the query; especially if it is a long query and the user is a 'one finger typist'.

The second solution could be a headache for the implementor but it is only done once: during the implementation. This approach is difficult for the user initially as two languages have to be learnt, but in time this will prove friendlier than the first approach.

Since this is a feasibility study and user acceptance is important, the second solution is the better of the two.

5.4.3.1 Types of the Editor

There are three common types of editors:

- Screen Editor
- Line Editor
- Text Editor

A screen editor is normally used with 'intelligent' terminals. These are normally cursor addressable VDUs. The difference between a Line Editor and a Text Editor is that in a Line Editor, edits may not be performed on lines preceding the current line within the same edit.

Of the three, a line editor is the easiest to design. The program given in Appendix 6 uses such an editor. The final implementation is not restricted to this choice.

5.4.3.2 Requirements of the Editor

A query is basically a very small data bank. This implies that the only transactions required will be

- Retrieve
- Update
- Insert
- Delete

It will also require a command to indicate the termination of an edit.

5.5 File Design

There is no fixed method of designing files. The common unorthodox methods which are used are the formation of hierarchies and the re-arrangement of shapes and lines until something solid arises!

The data and file structures were designed using both these techniques starting with the requirements of the basic system. In the description which follows, some line drawing will be done and some descriptions will be given in Algol 68. The two basic files which are required are

- The Data bank
- The Descriptor File

5.5.1 File Design for the Data bank

Each record has three conceptual parts:

```
MODE RECORD = STRUCT( PRIMARY key,
                      DETAILS source details,
                      TEST test results)
```

Each of these parts can be stored in separate files since their structures are different. Thus there will be three files:

- Primary Key File [PKF]
- Source Detail File [SDF]
- Test Results File [TRF]

5.5.1.1 The Primary Key File

The primary keys can be stored in the B-tree since they are unique and will only be 'consulted' when unique records are required. Each key could also hold a pointer to the position of the remaining details in the SDF. The structure of the PKF is given as follows:

```
MODE BTREE = STRUCT([0:order]REF BTREE branch,
                    [1:order] GROUP g),
     GROUP = STRUCT(PRIMARY key, INT source details)
```

5.5.1.2 Source Details File

Throughout the design, it has been assumed that the data bank is cumulative, that is, none of the records will be deleted. Since the secondary keys [or source details] have to be stored separately, the structure shown in Figure 5f can be adopted. Unfortunately, such a structure would restrict the number of records in the file. Reorganization would be inevitable if this number was exceeded.

Fortunately, this is not the only structure that can be adopted. Instead of presetting the space in a file, the file can be divided into blocks or pages [as they are called in jargonese]. Each page will contain a set of attributes [as described in Section 5.3.4.3] and will be linked to the next page containing the same set of attributes. Thus the file could be viewed as a series of linked pages [Figure 5g].

These pages need not be stored in order; page p need not contain attribute set (p mod (n+1)). They could be stored in any order but the computer would have to 'know' where each attribute started. This would be an advantage if a new set of attributes is added. The final structure is shown in Figure 5h.

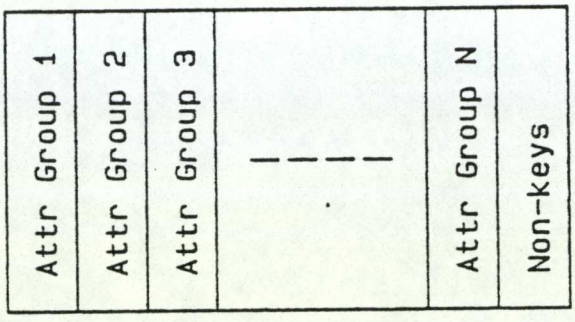


Figure 5f: Storage of Attribute Groups separately in a file. The space and number of records must be preset.

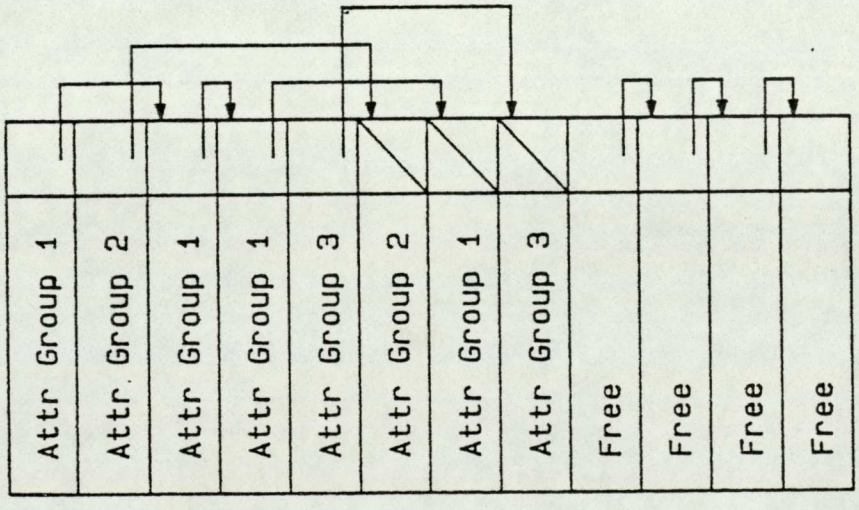


Figure 5g: Storage of Attribute Groups separately using linked pages. The advantage is that the space does not have to be preset. The disadvantage is that the system does not 'know' where each Attribute Group starts.

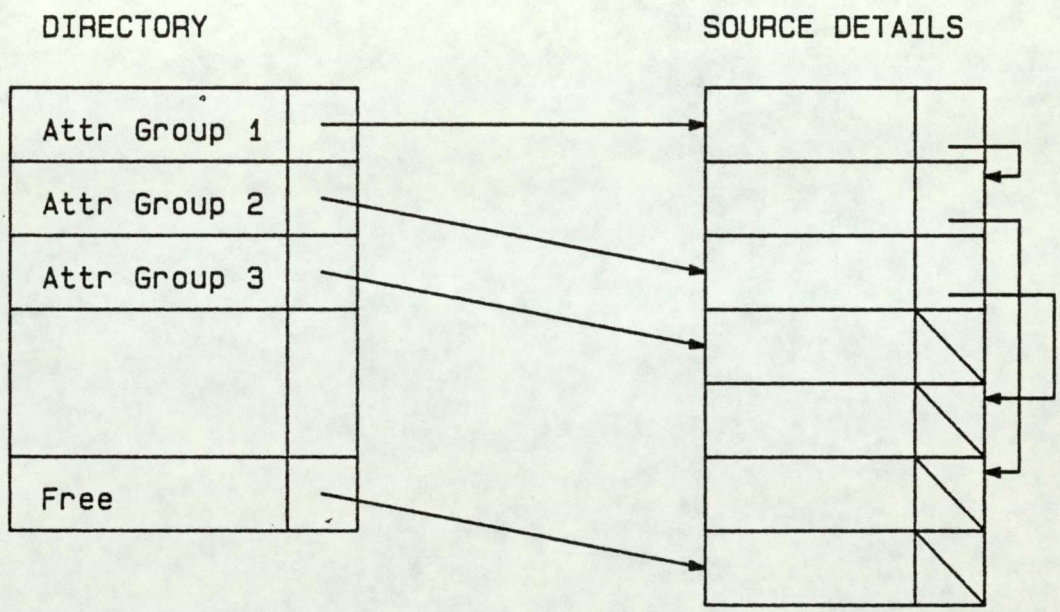


Figure 5h: Enhancement of Figure 5g. The directory shows where each page starts. Although the space will have to be preset for the directory, it will not cause as many problems as presetting for the number of records.

5.5.1.3 Test Results File

The test results present a similar problem to the source details. The number of tests would have to be fixed. This restricts the number of tests and wastes a lot of space by keeping it for future tests. This problem may be overcome by using links. The links can be stored as one of the attributes of the source details. The test results could be stored in the same way as the source details.

5.5.1.4 The Data Bank

Since B-trees also use pages [one node per page], the whole data bank can be stored as either one file or three files. This option is left open to the implementor. The final result is given in Figure 5i.

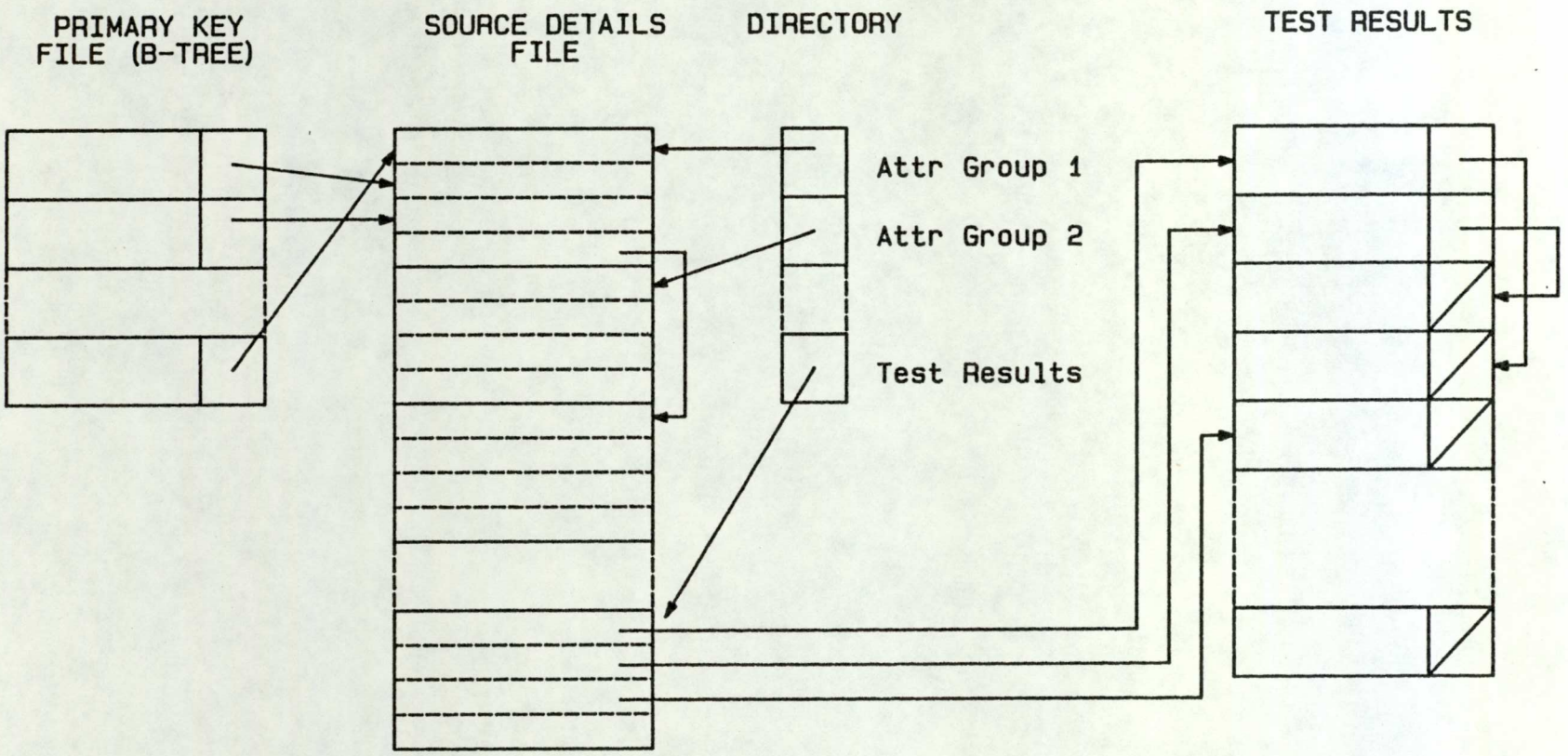


Figure 5i: Combined File Structure of the Data Bank

5.5.2 Design of the Descriptor File

The main aim of the descriptor file is to avoid the problem of having to recompile the program when a new attribute is added to the system. Each record has the same basic structure:

```
MODE RECORD  = STRUCT(STRING attribute, TYPE t,
                      INT no of elements, KEYTYPE k),
     TYPE    = UNION (INT, REAL, CODEDSTRING, STRING),
     KEYTYPE = UNION (PRIMARY, SECONDARY, NONKEY)
```

A CODEDSTRING is a set of strings with short forms. For instance, in a Chemical data bank, Mn could be used for manganese, Cl for chlorine, Na for sodium etc. A CODEDSTRING can be defined as

```
MODE CODEDSTRING = []STRUCT(STRING short form, long form)
```

By using CODEDSTRINGs, many typing errors can be avoided.

5.6 Implementation of the Information Retrieval System

Parts of the Information Retrieval System [IRS] were implemented on a PET in Basic. In the implementation, a variant of B-trees suggested by Martin [described by Knuth [1969b] but no references given] was used. The B-trees did not have to be of any specific order. Instead, each node was just left half full of data [which was variable in size]. The insertion and splitting mechanism still works even though the exact number of keys per node depends on the cumulative sizes of those keys.

A paging technique, based on the Least Recently Used [LRU] page replacement algorithm, was used for the search techniques. This method basically uses a fixed set of common buffers to contain pages of data read from the disk. When a page is required, a search is made for the next free buffer. If there are no free buffers then a search is made for the LRU buffer. This buffer is written to the disk and is set as the next free buffer. The required page is then read into this next free buffer. Using such a pool of common buffers has been found to be more efficient than using single buffers.

5.7 Discussion

In Section 5.6, it was mentioned that only parts of the system had been implemented on the PET. The main reason for this was to prove that the algorithms worked. They could have been developed on the Nova which was used for curve fitting but the program development would have been hindered due to the slow compilation speeds and 'unfriendly' error messges of the Nova. Moreover, many of the major routines would have to be rewritten for the object machine unless it was a Nova. These include all the character handling routines, the file handling routines and the implementation of the B-tree.

Further work could include checking both the syntax and validity of the queries. For instance,

age gt 5 and age lt 5

is not a valid query.

End of Chapter

CHAPTER 6: CONCLUSION

It is better to have a horrible ending than to
have horrors without end - Matsch's Law

It can be seen from the conclusions of Chapters 2, 3 and 5 that the whole project is by no means complete. In Chapter 2 [Data Capture], it was shown that the idea of using a 'microcomputer' for data collection was feasible. The extensions have already been mentioned and it is hoped that they will be implemented.

One of the problems envisaged during the data capture was the storage of the vast quantity of data. This led to a sub-project on Data Reduction [Chapter 3]. Even though the work is incomplete in this area, it identified the problems associated with data reduction. The identification of these problems could be useful to anyone wishing to pursue this area of research.

Although the Transfer of Data from the Data Capture Computer to the Data Reduction Computer [DRC] was manual, it would eventually have to be automated. This led to a sub-project on Data Transmission [Chapter 4]. It was by far the most successful of the sub-projects attempted. It was also the most useful as it was used to transmit the programs from the DRC to the Word Processor for reproduction in the Appendices.

The Data Storage and Retrieval sub-project [Chapter 5] was the collection point of the whole system. The reduced data, along with other relavant information could be held and retrieved from this system. The design of the system was interesting though only some sections were implemented due to the lack of a suitable machine. It showed the importance of choosing the correct machine to perform the task.

In conclusion, the whole project was a useful exercise in exploring the different areas in which machines could be used to make life easier for everyone. It showed the usefulness of general purpose tools and 'off the shelf' software for research work. Perhaps one day we will see the full Pulmonary Lung Function Breath Test Data Collection and Storage Equipment packaged in a box and available 'off the shelf' to any hospital!

End of Chapter

APPENDIX 1: ADC Programs and Output

The program listed in this Appendix was used to aquire data from the AIM 161 Analog to Digital Converter using a CBM 3032. For reasons of speed, one of the modules was written in Assembler. This is given in Appendix 1.1. The Basic driver is given in Appendix 1.2 and a sample of the printed output from a sinusoidal input is given in Appendices 1.3 and 1.4.

Note: Since the 'less than' and 'greater than' symbols were not avalilable on the printer, alphabetic replacements have been used. These are given as follows:

lt	less than
gt	greater than
ne	not equal to
ge	greater than or equal to
le	less than or equal to

In places where the substitution of these symbols leads to confusion, the actual wheel representations will be used. These are given as follows:

⅛	less than
⅜	greater than
£	hash symbol

This note applies to all the appendices.

Appendix 1.1: Assembler Program for Data Capture

```
.opt lis,nog,err
;       put "@0:adc
;       put "@1:adc
;
;       written: 25/02/80, cup
;       updated: 27/03/80, cup
;
;       purpose: analog to digital data aquisition via
;                a cmc aim 161 16-channel, 8 bit adc
;
;-----------------------------------------------------
       *=   826
;
; parameters of calling program
chanls =    251             ; number of channels
list   =    634             ; list of channels
sampls =    252             ; number of samples
t      =    2               ; delay time
;
; addresses for communications
cb2    =    59468
ieee   =    59426
usrprt =    59471
;
; local variables/constants
ptr    =    254
area   =    $55ff           ; recording area
sslo   =    %11011111       ; start strobing off
sshi   =    %00100000       ; start strobing on
;
; initialize
       sei
       cld
       inc  sampls+1
       lda £$ff             ; put recording address in ptr
       ldx £$55             ; high byte
       sta ptr
       stx ptr+1
       lda £sshi            ; stop strobing
       ora cb2
       sta cb2
nexsam = *
       ldy chanls           ; get ready to receive data
nextch = *
       lda list-1,y         ; select channel number
       sta ieee
       lda £sslo            ; start strobing
       and cb2
       sta cb2
       lda £$40             ; tell adc to flag eoc
       ora ieee
       sta ieee
```

```
        ldx t               ; wait for t*5+87 microseconds
wait    = *
        dex
        bne wait
eoc     = *
        lda usrprt          ; check for eoc just in case
        bpl eoc             ; wait is too short
        lda £sshi           ; put converted data in user port
        ora cb2
        sta cb2
        lda usrprt          ; read converted data
        sta (ptr),y
        dey
        bne nextch
        dec sampls          ; end of run?
        bne nexblk
        dec sampls+1
        bne nexblk
        cli
        rts
nexblk = *
        clc                 ; move pointer to next block
        lda chanls
        adc ptr
        sta ptr
        bcc nexsam
        inc ptr+1
        jmp nexsam
        .end
```

Appendix 1.2: Basic Driver

```
10000  REM===ADC
10010  REM  AUTHOR       : CUP
10020  REM  DATE WRITTEN: 01/02/80
10030  REM  INSTALLATION: DEPT PTPM ,ASTON UNIVERSITY,BIRMINGHAM
10040  REM                C.I.UNIT,DUDLEY ROAD HOSPITAL,BIRMINGHAM
10080  PRINT "[scr.clr][rvs.on]Data Aquisition from an ADC"
10090  PRINT TAB(9)"[rvs.on]Last Update: 02/04/80"

10100  REM===VARIABLES & CONSTANTS
10101  AX%=0:
       AY%=0:
       A$=""
10102  B$=" "
10103  C$="":
       C=0:
       CA=0:
       CB=59468:
       CH=0:
       CV=826:
       CT$=""
10104  D=0:
       DC=0:
       DE=0:
       DV=4
10107  GP=32256
10108  H=100:
       HC=0:
       HI=0:
       HR=1E38
10109  I=0:
       IE=59426
10110  J=0
10111  K=0
10112  L=0:
       LI=0:
       LO=0:
       LR=-1E38
10113  MC$="m/c code"
10114  N=0:
       NA=0:
       NC=0
10116  P$="":
       PR=0:
       PT=32512
10119  S$="":
       SM=50:
       SK=0:
       SP=30208:
       SR=22016:
       SS=100
10120  T=0:
       TB=0:
       TT=0
```

```
10121  UP=59471
10124  XC=0
10125  YC=0

10200  REM===ARRAYS
10202  DIM BA(16),BV(16)
10203  DIM CH$(6),CN(17),CL(17)
10214  DIM NM$(17)
10218  DIM RV(17)
10219  DIM SC(17)
10221  DIM UN$(17)

10300  REM===FUNCTIONS
10319  DEF FNS(V)=BA(CH)+SC(CH)*(V-BV(CH))

10400  REM===MAIN PROGRAM

10410  REM--INITIALIZE
       GOSUB 11000:

10420  REM--CALIBRATE
       GOSUB 13000:

10430  REM--DISPLAY MENU
       GOSUB 12000:
10440  ON N GOSUB 13000,14000,15000,16000,17000,18000
10450  GOTO 10430

11000  REM===INITIALIZE

11100  REM--LOWER CASE
       POKE 59468,14:

11110  REM--SET MEMORY BOUNDS
11120  N=INT(SR/256)-1:
       POKE 49,N:
       POKE 51,N:
       POKE 53,N
11130  PRINT "[cur.dwn]Have the machine code routines been
       loaded";:
       GOSUB 33000
11140  IF S$="y" THEN 11300

11200  REM--LOAD M/C CODE PROCS
11210  PRINT "[cur.dwn]Are you loading from tape";:
       GOSUB 33000
11220  D=1:
       DC=0:
       IF S$="y" THEN 11250
11230  D=8:
       DC=2:
       PRINT "[cur.dwn]Drive number";:
       LO=0:
       HI=1:
       GOSUB 30000:
```

```
        OPEN 1,8,15,"i"+CHR$(48+N)
11240   MC$=CHR$(48+N)+":"+MC$+",s,r"
11250   OPEN 2,D,DC,MC$:
        GOSUB 36000
11260   INPUT £2,L,C:
        IF L gt 0 THEN
           POKE L,C:
           GOTO 11260
11270   CLOSE 2:
        IF D=8 THEN
           CLOSE 1

11300   REM--MAIN MENU
11310   DATA "Calibration and Selection of Channels"
11320   DATA "Data Aquisition"
11330   DATA "Tabulation of Aquired Data"
11340   DATA "Graphs of Aquired Data"
11350   DATA "Dump of Aquired Data"
11360   DATA "Exit from System"
11390   FOR I=1 TO 6:
           READ CH$(I):
        NEXT I
11400   REM--OTHER STRINGS
11410   NM$(16)="Time":
        UN$(16)="millisecs"
11500   FOR I=1 TO 5:
           B$=B$+B$:
        NEXT I:
        B$=LEFT$(B$+B$,38)

11600   REM--CHECK IF ADC IS CONNECTED
11610   PRINT "[cur.dwn]Is the ADC connected";:
        GOSUB 33000:
        NA=(S$="n")
11999   RETURN

12000   REM===MAIN MENU
12010   PRINT "[scr.clr]"TAB(12)"[rvs.on]Main Menu
        [rvs.off][4cur.dwn]"
12100   FOR I=1 TO 6:
           PRINT "[cur.dwn][rvs.on]"CHR$(I+48)"[rvs.off] "CH$(I):
        NEXT I
12110   PRINT "[2cur.dwn]  Option";:
        LO=1:
        HI=6:
        GOSUB 30000
12999   RETURN

13000   REM===CALIBRATION
13010   PRINT "[scr.clr]"TAB(15)"[rvs.on]Calibration[2cur.dwn]"
13015   REM--DISPLAY CHANNELS IN USE
13020   FOR I=0 TO 15
13030      S$=RIGHT$(STR$(I),2):
           IF I lt 10 THEN
              S$=RIGHT$(S$,1)
```

```
13040      IF NM$(I)="" THEN
              PRINT " ";S$;
13050      IF NM$(I) ne "" THEN
              PRINT " [rvs.on]";S$;"[rvs.off]";
13060   NEXT I
13070   PRINT

13080   REM--OBTAIN INFO ABOUT CHANNEL TO BE USED/DELETED
13100   PRINT "[cur.dwn]Channel";:
        LO=0:
        HI=15:
        GOSUB 30000:
        CH=N
13110   IF NM$(CH)="" THEN 13200
13120   PRINT "[cur.dwn]This channel has already been taken"
13130   PRINT "for [rvs.on]"NM$(CH)"[rvs.off]."
13140   PRINT "[cur.dwn]Do you wish to clear it";:
        GOSUB 33000
13150   IF S$="y" THEN
           NM$(CH)="":
           GOTO 13800
13160   PRINT "[cur.up]Do you wish to change it";:
        GOSUB 33000:
        IF S$="n" THEN 13800
13170   PRINT "[4cur.up]"B$:
        PRINT B$:
        PRINT "[cur.dwn]"B$"[5cur.up]"
13190   PRINT "[cur.dwn]"B$"[5cur.up]"
13200   PRINT "[cur.dwn]Title";:
        LO=1:
        HI=16:
        GOSUB 34000:
        NM$(CH)=S$
13210   PRINT "[cur.dwn]Units";:
        LO=1:
        HI=16:
        GOSUB 34000:
        UN$(CH)=S$
13220   PRINT "[cur.dwn]Lower calibration value";:
        LO=LR:
        HI=HR:
        GOSUB 30000:
        BA(CH)=N
13230   GOSUB 38000:
        BV(CH)=N
13240   PRINT "[scr.home][13cur.dwn]Higher calibration value";
13250   LO=LR:
        HI=HR:
        GOSUB 30000:
        HC=N
13260   GOSUB 38000

13270   REM--FIND THE SCALE
13280   SC(CH)=(HC-BA(CH))/(N-BV(CH))
```

```
13800  REM--DISPLAY CHANNELS
13810  PRINT "[scr.clr][cur.dwn][rvs.on]Channels in use[cur.dwn]":
       NC=0
13820  FOR I=0 TO 15
13830     IF NM$(I) ne "" THEN
             PRINT I;TAB(4);NM$(I)"("UN$(I)")":
             NC=NC+1
13840  NEXT I

13845  REM--DECIDE WHETHER TO TERMINATE OR TO CONTINUE
13850  IF NC ne 0 THEN 13900
13860  PRINT "[scr.clr][11cur.dwn][rvs.on]"TAB(11)"No channels in
       use":
       GOSUB 59000:
       GOTO 13000
13900  PRINT "[scr.home][21cur.dwn]Anymore";:
       GOSUB 33000:
       IF S$="y" THEN 13000
13999  RETURN

14000  REM===CONVERSION
14010  PRINT "[scr.clr]"TAB(15)"[rvs.on]Conversion":
       IF NC=0 THEN 50000
14020  NC=ABS(NC)

14025  REM--OBTAIN DETAILS ABOUT THE SAMPLING RATE
14030  LO=NC*0.2:
       HI=NC:
       PRINT "[2cur.dwn]Period in between samples for the"
14040  PRINT "same channel in milliseconds"
14045  PRINT "(between"LO" and"HI")";:
       GOSUB 30000:
       SS=N
14050  LO=2:
       HI=INT(8192/NC):
       PRINT "[2cur.dwn]Number of Samples (less than "HI")";
14060  GOSUB 30000:
       SM=N
14070  DE=INT((SS*1E3/NC-87)*0.2)+1:
       N=SS:
       SS=((DE-1)*5+87)*NC*1E-3
14075  IF N=SS THEN 14100
14080  PRINT "The actual period in betewen samples"
14090  PRINT "for the same channel will be"SS

14100  REM--SET UP FOR CONVERSION
14110  POKE 251,NC:
       POKE 2,DE:
       POKE 252,SM AND 255:
       POKE 253,INT(SM/256)

14200  REM--SET THE CHANNELS
       CA=634:
14210  FOR I=0 TO 15
```

```
14220     IF NM$(I) ne "" THEN
             POKE CA,I:
             CA=CA+1
14230  NEXT I
14300  GOSUB 35000:
       IF NOT NA THEN
          SYS (CV)
14310  IF NC=1 THEN 14600

14400  REM--INTERPOLATE SO THAT ALL THE READINGS APPEAR TO BE

14410  REM--TAKEN AT THE SAME TIME
14420  CA=SR:
       T=NC-2
14430  FOR I=1 TO SM
14440     PR=CA:
          CA=CA+NC:
          D=NC
14450     FOR J=T TO 0 STEP -1
14460        D=D-1:
             N=PEEK(PR+J):
             N=N+INT(((PEEK(CA+J)-N)/NC)*D):
             POKE PR+J,N
14500     NEXT J
14510  NEXT I
14600  NC=-NC
14999  RETURN

15000  REM===TABULAR OUTPUT
15010  PRINT "[scr.clr]"TAB(12);"[rvs.on]Tabular Output"
15020  IF NC=0 THEN 50000
15030  IF NC gt 0 THEN 51000
15040  NC=-NC:
       LI=-1:
       PRINT "[2cur.dwn]":
       NM$(17)="End of Table"

15045  REM--DISPLAY CHANNELS IN USE
15050  FOR I=0 TO 17
15060     IF NM$(I) ne "" THEN
             PRINT I;TAB(4)NM$(I):
             LI=LI+1:
             RV(I)=LI
15070  NEXT I

15080  REM--OBTAIN THE CONTENTS OF EACH COLUMN
15100  FOR I=1 TO 5
15110     P$="Column"+STR$(I):
          GOSUB 37000
15120     IF N=17 THEN 15300
15200  NEXT I
15300  CA=SR:
       TB=I-1
15310  IF TB=0 THEN 15600
15315  OPEN 1,DV,1:
```

```
      TT=0

15317 REM--PRINT THE HEADING
15320 A$="":
      FOR I=1 TO TB:
         A$=A$+RIGHT$(B$+NM$(CN(I)),15):
      NEXT I:
      SYS (PT):
      PRINT £1,A$
15330 FOR I=1 TO SM
15340    S$=""

15345    REM--PRINT THE CONVERTED RESULTS
15350    FOR J=1 TO TB
15360       CH=CN(J):
            IF CH=16 THEN
               S$=S$+RIGHT$(B$+STR$(TT),15)
15370       IF CH ne 16 THEN
               S$=S$+RIGHT$(B$+STR$(FNS(PEEK(CA+CL(J)))),15)
15380    NEXT J
15390    TT=TT+SS:
         CA=CA+NC:
         PRINT £1,S$
15400 NEXT I
15410 PRINT £1,CHR$(12)
15420 CLOSE 1
15500 IF DV=3 THEN
         GOSUB 35000
15600 NC=-NC
15999 RETURN

16000 REM===GRAPHICAL OUTPUT
16010 PRINT "[scr.clr]"TAB(12)"[rvs.on]Graphical Output"
16020 IF NC=0 THEN 50000
16030 IF NC gt 0 THEN 52000

16100 REM--DISPLAY THE CHANNELS
16110 NC=-NC:
      LI=-1:
      PRINT "[2cur.dwn]":
      NM$(17)="Return to Main Menu"
16120 FOR I=0 TO 17
16130    IF NM$(I) ne "" THEN
            PRINT I;TAB(4);NM$(I):
            LI=LI+1:
            RV(I)=LI
16140 NEXT I

16145 REM--OBTAIN X AND Y AXES
16150 I=1:
      P$="X-Axis":
      GOSUB 37000:
      AX%=CL(I):
      XC=N:
      IF N=17 THEN 16800
```

```
16160  I=2:
       P$="Y-Axis":
       GOSUB 37000:
       AY%=CL(I):
       YC=N:
       IF N=17 THEN 16800

16300  REM--INITIALIZE 'PLOTTER'
16310  OPEN 1,DV,1:
       FOR I=1 TO 6:
          PRINT £1:
       NEXT I

16400  REM--'DRAW' GRAPH

16410  REM--ADDR OF A$
       CH=YC:
       CA=SR:
       TT=SS*SM:
       N=TT/8:
       I=PEEK(42)+PEEK(43)*256+14:
16420  FOR CA=SP TO SP+2047 STEP 64
16430     POKE I+2,64:
          POKE I+3,CA AND 255:
          POKE I+4,INT(CA/256)
16440     S$=LEFT$(B$,15):
          T=(CA-SP)/64

16445     REM--INSERT SCALE VALUES
16450     IF LEFT$(A$,1) ne "+" THEN 16460
16455     IF CH ne 16 THEN
             S$=RIGHT$(B$+STR$(FNS(248-T*8)),15)
16456     IF CH=16 THEN
             TT=TT-N:
             S$=RIGHT$(B$+STR$(TT),15)
16460     S$=S$+A$:
          PRINT £1,S$
16470  NEXT CA

16525  REM--SCALE VALUES ON X AXIS
16530  CH=XC:
       S$="":
       N=SS*SM/4:
       TT=0
16540  FOR CA=0 TO 256 STEP 64
16550     IF CH ne 16 THEN
             S$=S$+RIGHT$(B$+STR$(FNS(CA)),16)
16555     IF CH=16 THEN
             S$=S$+RIGHT$(B$+STR$(TT),16):
             TT=TT+N
16560  NEXT CA
16570  PRINT £1,S$:
       FOR I=1 TO 5:
          PRINT £1:
       NEXT
```

```
16600  REM--TITLE
16610  A$=LEFT$(B$,16)+"X-axis = "+NM$(XC)+"("+UN$(XC)+")":
       SYS (PT):
       PRINT £1,A$
16620  A$=LEFT$(B$,16)+"Y-axis = "+NM$(YC)+"("+UN$(YC)+")":
       SYS (PT):
       PRINT £1,A$
16630  PRINT £1:
       A$=LEFT$(B$,16)+CT$:
       SYS (PT):
       PRINT £1,A$
16640  PRINT £1,CHR$(12)
16700  CLOSE 1:
       IF DV=3 THEN
          GOSUB 35000
16800  NC=-NC
16999  RETURN

17000  REM===SAVE RESULTS
17010  PRINT "[scr.clr]"TAB(9)"[rvs.on]Dump Converted Results"
17020  IF NC=0 THEN 50000
17030  IF NC gt 0 THEN 53000
17040  NC=ABS(NC)
17050  PRINT "[2cur.dwn]Devices Available":
       PRINT "[17£]"
17070  PRINT "[rvs.on]1[rvs.off] Tape Drive"
17080  PRINT "[rvs.on]2[rvs.off] Disk Drive 0"
17090  PRINT "[rvs.on]3[rvs.off] Disk Drive 1"
17100  PRINT "[cur.dwn]Device to be used";:
       LO=1:
       HI=3:
       GOSUB 30000
17110  PRINT "[cur.dwn]Filename";:
       LO=1:
       HI=16:
       GOSUB 34000:
       F$=S$
17120  D=4:
       DC=1:
       IF N gt 1 THEN
          D=8:
          DC=2:
          F$="@"+CHR$(46+N)+":"+F$:
          OPEN 1,8,15
17140  OPEN 2,D,DC,F$

17200  REM--QUANTITIES
17210  PRINT £2,CT$;CR$;NC;CR$;SM;CR$;SS;CR$;
```

```
17300  REM--NAMES & UNITS
17310  J=0
17320  FOR I=0 TO 15
17330     IF NM$(I) ne "" THEN
             J=J+1:
             PRINT £2,NM$(I);CR$;UN$(I);CR$;:
             CN(J)=I
17340  NEXT I

17400  REM--RAW RESULTS
17420  FOR I=1 TO SM
17430     FOR J=1 TO NC
17440        PRINT £2,Z$;CHR$(PEEK(CA+J));
17460     NEXT J
17470     CA=CA+NC
17480  NEXT I
17490  CLOSE 2:
       IF D=8 THEN
          CLOSE 1
17510  NC=-NC
17999  RETURN

18000  REM===EXIT FROM SYSTEM
18010  PRINT "[scr.clr][10cur.dwn]"TAB(18)"[rvs.on]bye[10cur.dwn]"
18999  END

30000  REM===GET A NUMBER
30005  PRINT "? ";
30010  S$=""
30020  GOSUB 31000
30030  IF C=20 THEN
          GOSUB 32000:
          GOTO 30020
30040  IF C=13 AND S$ ne "" THEN 30100
30050  IF (C$ ge "0" AND C$ le "9") OR C$="." OR C$="-" OR C$="e"
THEN
          S$=S$+C$:
          PRINT C$;
30060  GOTO 30020
30100  N=VAL(S$):
       IF N ge LO AND N le HI THEN
          PRINT " ":
          RETURN
30110  FOR L=1 TO LEN(S$):
          PRINT " [2cur.lft] [cur.lft]";:
       NEXT L
30120  GOTO 30010

31000  REM===GET A CHARACTER
31010  GET C$:
       IF C$ ne "" THEN 31100
```

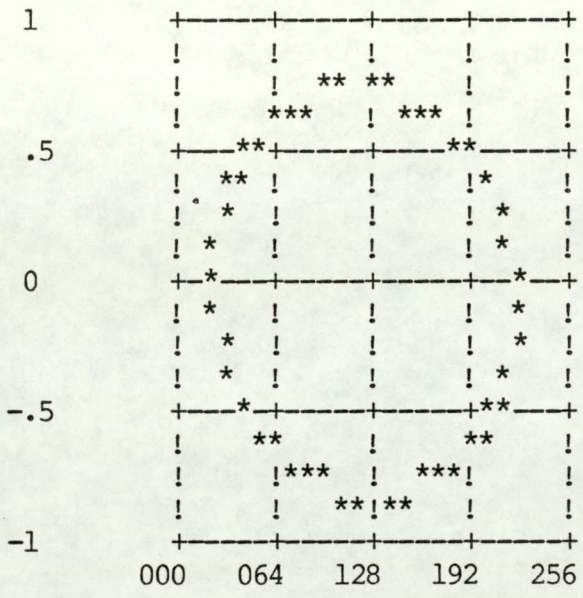
```
31020  PRINT "*[cur.lft]";:
       FOR K=1 TO H:
       NEXT K
31030  GET C$:
       IF C$ ne "" THEN 31100
31040  PRINT "+[cur.lft]";:
       FOR K=1 TO H:
       NEXT K
31050  GOTO 31000
31100  C=ASC(C$)
31999  RETURN

32000  REM===DELETE A CHARACTER
32010  L=LEN(S$):
       IF L=0 THEN
          RETURN
32020  PRINT " [2cur.lft] [cur.lft]";
32030  S$=LEFT$(S$,L-1)
32999  RETURN

33000  REM===GET Y OR N
33005  PRINT "? ";
33010  GOSUB 31000
33020  IF C$="N" OR C$="Y" THEN
          C$=CHR$(ASC(C$)-128)
33030  IF C$ ne "y" AND C$ ne "n" THEN 33010
33040  PRINT C$;:
       S$=C$
33100  GOSUB 31000
33110  IF C=20 THEN
          PRINT " [2cur.lft] [cur.lft]";:
          GOTO 33010
33120  IF C ne 13 THEN 33100
33130  PRINT " "
33999  RETURN

34000  REM===GET STRING
34010  PRINT "? ";
34020  S$=""
34030  GOSUB 31000
34040  IF C=20 THEN
          GOSUB 32000:
          GOTO 34030
34050  IF C=13 THEN 34100
34060  IF (C ge 32 AND C le 95) OR (C ge 192 AND C le 222) THEN
          S$=S$+C$:
          PRINT C$;
34070  IF C=34 THEN
          PRINT C$"[cur.lft] [cur.lft]";
34080  GOTO 34030
34100  L=LEN(S$):
       IF L ge LO AND L le HI THEN
          PRINT " ":
          RETURN
```

```
34110  IF L ne 0 THEN
          FOR L=1 TO LEN(S$):
             PRINT " [2cur.lft] [cur.lft]";:
          NEXT L
34120  GOTO 34020

35000  REM===WAIT
35010  PRINT "[scr.home][23cur.dwn][rvs.on]Strike any key when
       ready[rvs.off]"
35020  GET C$:
       IF C$="" THEN 35020
35030  PRINT "[cur.up]"B$
35999  RETURN

36000  REM===CHECK DISK ERRORS
36010  IF DC ne 2 THEN
          RETURN
36020  INPUT £1,ER,EM$,T,S
36030  IF ER=0 THEN
          RETURN
36040  PRINT "Disk error:"EM$
36050  CLOSE 1
36060  GOSUB 35000
36999  RETURN

37000  REM===GET A CHANNEL
37010  PRINT "[scr.home][20cur.dwn]"P$"          [9cur.lft]";
37020  LO=0:
       HI=17:
       GOSUB 30000
37030  IF N=17 THEN
          RETURN
37040  IF NM$(N)="" THEN 37200
37100  CL(I)=RV(N):
       CN(I)=N

37110  REM--HILITE CHANNEL
37120  PRINT "[scr.home][2cur.dwn]":
       FOR J=1 TO RV(N)+1:
          PRINT :
       NEXT J
37130  PRINT "[rvs.on]"N"[rvs.off]"TAB(4)NM$(I)
37140  RETURN
37200  PRINT "[cur.dwn][rvs.on]No such option"
37210  FOR K=1 TO 2000:
       NEXT K
37220  PRINT "[cur.up]"LEFT$(B$,31):
       GOTO 37000

38000  REM===CALIBRATE THE INPUT
38010  IF NA THEN
          PRINT "Mean Input Voltage";:
          LO=0:
          HI=255:
          GOSUB 30000:
```

```
          RETURN

38020  REM--NO OF SAMPLES
       SM=256:
38060  GOSUB 35000:
       N=0

38100  REM--FIND THE MEAN
38110  FOR I=1 TO SM
38120     POKE CB,238
38130     POKE IE,CH
38140     POKE CB,206
38150     J=PEEK(IE):
          POKE IE,J OR 64
38160     POKE CB,238
38170     N=N+PEEK(UP):
          PRINT "[cur.up]"PEEK(UP)
38180  NEXT I
38190  N=N/SM:
       PRINT "[cur.up]Mean value is"N
38200  FOR I=1 TO 100:
       NEXT I:
       PRINT "[cur.up]"LEFT$(B$,39)
38999  RETURN

50000  REM===ERROR MESSAGES
50010  PRINT TAB(8)"[2cur.dwn]You have not selected or"
50020  PRINT TAB(6)"calibrated any channels yet":
       GOTO 59000
51000  PRINT TAB(6)"[2cur.dwn]There is nothing to tabulate":
       GOTO 59000
52000  PRINT TAB(8)"[2cur.dwn]There is nothing to plot":
       GOTO 59000
53000  PRINT TAB(8)"[2cur.dwn]There is nothing to dump"

59000  REM===HOLD ERROR MESSAGE
59010  FOR I=1 TO 1500:
       NEXT I
59020  RETURN
```

Appendix 1.3 Sample Graphical Output

The output given here is the result of a later program which had been substantially modified to give outputs on disk and tape instead of the printer or screen. The results have had a long journey; having been transferred first from the CBM 3040 disk unit at Dudley Road to a CBM C50 cassette unit, then from a CBM C50 to the 400K Compu/Think disk unit at Aston and finally from the 400K Compu/Think to a CBM 8050 Disk Unit from where it was loaded on to the word processor to be output in this appendix.



X axis: Oscillator()
Y axis: Signal Generator()

Appendix 1.4 Sample Tabular Output

These are the results which were stored with the graph given in Appendix 1.3. The heading given at the beginning of each column is restricted to 15 characters, hence the truncated 'Signal Generator'.

Osciallator	Signal Generato
128	.7734375
152	.7421875
176	.65625
196	.53125
212	.375
223	.1875
227	0
226	-.1953125
218	-.3828125
205	-.5390625
186	-.6640625
164	-.75
140	-.78125
115	-.7734375
91	-.7109375
69	-.609375
50	-.4609375
37	-.2890625
29	-.1015625
28	.09375
32	.28125
43	.453125
59	.6015625
79	.703125
103	.765625

End of Appendix

APPENDIX 2

The programs given in this appendix may be used for curve fitting in general. Two basis functions are used:

- Chebyschev Polynomials
- Orthogonal Polynomials

The method which uses Chebyschev polynomials is based on the Minimax norm whilst the method of Orthogonal polynomials is based on the Least Squares norm. Both these programs are Fortran translations of Algol 60 programs. They utilize some of the special features provided by Data General; for instance, PARAMETER statements. If they are to be transferred to any other machine other than a Data General, it is advisable to rewrite them in Fortran 77 or Fortran 8X [when X3J3 decide to release it!]

Appendix 2.1

```
C
C     Written: 21/01/81, CUP
C
C     Chebyschev Curve Fit (Modified) [J. Boothroyd]
C     Algorithm 318, CACM 10:12, 801-803, 1967
C
C     Evaluates in COEF(0..ORDRQD) the coefficients of an ORDRQD
C     polynomial such that the maximum error is minimum over the
C     sample points (NUMPTS .GT. ORDRQD+1)
C
C     The X values must form a strict monotonic sequence.
C
C     With finite precision arithmetic, this procedure will not
C     always terminate after a finite number of steps. Roundoff
C     errors may cause looping of the chosen reference sets. This
C     ill-fated condition is detected by checking that the
C     reference deviation is always raise monotonically. On exit,
C     the absolute value of COEF(ORDPL1) yields the final
C     reference deviations. Negative COEF(ORDPL1) indicates that
C     the procedure has been terminated following the detection of
C     cycling.
C
      COMPILER DOUBLE PRECISION
      PARAMETER MAXORD = 10, MAXOR1 = 11
C
      SUBROUTINE CURFIT( XMIN,XMAX,Y,START,LSTPT,COEF,ORDPL1 )
      INTEGER LSTPT, ORDPL1, START
      REAL    COEF(0:ORDPL1), XMIN, XMAX, Y
C
      INTEGER I, I1, IMAX, J, J1, K, NUMPTS, ORDRQD
      INTEGER REF(0:MAXOR1), REFI, REFJ
      REAL    ABSHI, COEFI, COEFI1, DENOM, H, HMAX, HI, HIMAX
      REAL    NEXTHI, PREVH, POSN, POSDIF, REFHI, REFHI1
      REAL    REFX(0:MAXOR1), REFH(0:MAXOR1), STEP, XJ
C
      EXTERNAL OBTAIN, RESET
C
      NUMPTS = LSTPT - START + 1
      PREVH  = 0.0
      ORDRQD = ORDPL1 - 1
      STEP   = ( XMAX - XMIN ) / FLOAT( NUMPTS )
C
C     Index vector for initial reference set
      REF(0) = START
      REF(ORDPL1) = LSTPT
      POSDIF = FLOAT( NUMPTS - 1 ) / FLOAT( ORDPL1 )
      DO 100 I = 1, ORDRQD, 1
         REF(I) = INT( POSN )
         POSN   = POSN + POSDIF
100   CONTINUE
```

```
C
C     Start of iterative loop
110   CONTINUE
         H = -1.0
C
C        Select ORDRQD+2 reference pairs and set alternating
C        deviation vector
         DO 120 I = 0, ORDPL1, 1
            REFI = REF(I)
            REFX(I) = XMIN + FLOAT( REFI - START ) * STEP
            COEF(I) = Y(REFI)
            H = -H
            REFH(I) = H
120      CONTINUE
C
C        Compute ORDPL1 leading dividend differences
         DO 140  J = 0, ORDRQD, 1
            I1 = ORDPL1
            COEFI1 = COEF(I1)
            REFHI1 = REFH(I1)
            I = ORDRQD
130         CONTINUE
               DENOM = REFX(I1) - REFX(I - J)
               COEFI = COEF(I)
               REFHI = REFH(I)
               COEF(I1) = ( COEFI1 - COEFI ) / DENOM
               REFH(I1) = ( REFHI1 - REFHI ) / DENOM
               I1 = I
               COEFI1 = COEFI
               REFHI1 = REFHI
               I = I - 1
               IF ( I .GE. J ) GOTO 130
140      CONTINUE
C
C        Equate the ORDPL1th difference to zero to determine H
         H = - COEF(ORDPL1)  / REFH(ORDPL1)
C
C        With H known, combine the function and deviation
C        differences
         DO 150 I = 0, ORDPL1, 1
            COEF(I) - COEF(I) + REFH(I) * H
150      CONTINUE
C
C        Polynomial coefficients
         J = ORDRQD - 1
160      CONTINUE
            XJ = REFX(J)
            I = J
            COEFI = COEF(I)
            K =  J + 1
            DO 170 I1 = K, ORDRQD, 1
               COEFI1 = COEF(I1)
               COEF(I) = COEFI - XJ * COEF(I1)
               COEFI = COEFI1
               I = I1
```

```
170         CONTINUE
          J = J - 1
          IF ( J .GE. 0 ) GOTO 160
C
C         Terminate if the reference deviation is not
C         increasing monotonically
          HMAX = ABS( H )
          IF ( HMAX .GT. PREVH ) GOTO 180
             COEF(ORDPL1) = -HMAX
             RETURN
180       CONTINUE
C
C         Find the index, IMAX  and the value HIMAX of the largest
C         absolute error for all the sample points
          PREVH = HMAX
          COEF(ORDPL1) = HMAX
          IMAX = REF(0)
          HIMAX = H
          J = 0
          REFJ = REF(J)
          DO 240 I = START, LSTPT, 1
             X = XMIN + FLOAT( I - START ) * STEP
             IF ( I .EQ. REFJ ) GOTO 210
                HI = COEF(ORDRQD)
                K = ORDRQD - 1
190             CONTINUE
                   HI = HI * X + COEF(K)
                   K = K - 1
                   IF ( K .GE. 0 ) GOTO 190
                HI = HI - Y(I)
                ABSHI = ABS( HI )
                IF ( ABSHI .LE. HMAX ) GOTO 200
                   HMAX = ABSHI
                   HIMAX = HI
                   IMAX = I
200             CONTINUE
                GOTO 230
210          CONTINUE
                IF ( J .GT. ORDPL1 ) GOTO 220
                   J = J + 1
                   REFJ = REF(J)
220             CONTINUE
230          CONTINUE
240       CONTINUE
C
C         If the maximum error occurs at a non-reference point,
C         exchange this point with the nearest reference point
C         having the same sign and repeat
          IF ( IMAX .EQ. REF(0) ) RETURN
          DO 250 I = 0, ORDPL1, 1
             IF ( IMAX .LT. REF(I) ) GOTO 260
250       CONTINUE
          I = ORDPL1
```

```
C
C       Swap
260     CONTINUE
        NEXTHI = SIGN( H, 0.5 - FLOAT( MOD( I, 2 ) ) )
        IF ( HIMAX * NEXTHI .LT. 0 ) GOTO 290
           REF(I) = IMAX
           GOTO 340
290     CONTINUE
           IF ( IMAX .GE. REF(0) ) GOTO 310
              J1 = ORDPL1
              J = ORDROD
300           CONTINUE
                 REF(J1) = REF(J)
                 J1 = J
                 J = J - 1
                 IF ( J .GE. 0 ) GOTO 300
              REF(0) = IMAX
              GOTO 340
310        CONTINUE
           IF ( IMAX .LE. REF(ORDPL1) ) GOTO 330
              J = 0
              DO 320 J1 = 1, ORDPL1, 1
                 REF(J) = REF(J1)
                 J = J1
320           CONTINUE
              REF(ORDPL1) = IMAX
              GOTO 340
330        CONTINUE
              REF(I - 1) = IMAX
340     CONTINUE
        GOTO 110
      END
```

Appendix 2.2

```
C
C     Written: 03/06/81, CUP
C
C     Performs a Least Squares Fit.  COEF(0..ORDRQD) are the
C     coefficients of the 'best' polynomial approximation of
C     degree ORDRQD or less as programmed by the method of
C     orthogonal polynomials described by G.E. Forsythe in Journal
C     SIAM 5:2, 1957 (with minor alterations).
C
C     The calculations are performed in the range -2 to 2; XMIN
C     and XMAX will be scaled accordingly before and after the
C     curve fitting routine.
C
C     This subroutine was developed from the programs 'Least
C     Squares Fit by Orthogonal Polynomials' and 'Polynomial
C     Transformer' due to MacKinney, CACM 3, 604 after
C     implementing the remarks made by MacMillan, CACM 4, 544 and
C     Makinson, CACM 10, 293.
C
      COMPILER DOUBLE PRECISION
      PARAMETER MXORD = 10, MXPTS = 500
C
      SUBROUTINE CURFIT( XMIN,XMAX,Y,START,LSTPT,COEF,ORDPL1 )
      INTEGER LSTPT, ORDPL1, START
      REAL    COEF(0:ORDPL1), Y(1:LSTPT)
C
      INTEGER DEG, LSTORD, NUMPTS, ORD, ORDP1, ORDRQD, POLYNO, PT
      LOGICAL SWX
      REAL    CONST, DUMMY, SCALE, SORSQC, SORSQP, SUMOTH
      REAL    X, XINC, XINCSQ, XZER
      REAL    ALPHA(0:MXORD), BETA(0:MXORD), CFORCR(0:MXORD)
      REAL    CFORPV(-1:MXORD), CFORPY(0:MXORD), CONPWR(0:MXORD)
      REAL    CRORVL(1:MXPTS), PVORVL(1:MXPTS), SCLFAC(0:MXORD)
      REAL    SMOTH2(0:MXORD), SUMSQ(0:MXORD)
C
C     Initialization
      ORDRQD = ORDPL1 - 1
      SWX = .TRUE.
      BETA(0) = 0.0
      CFORPV(-1) = 0.0
      CFORPV( 0) = 0.0
      CFORCR( 0) = 1.0
      XINCSQ = 0.0
      SUMOTH = 0.0
      NUMPTS = LSTPT - START + 1
      SORSQC = FLOAT( NUMPTS )
C
      DO 100 PT = START, LSTPT
         XINCSQ = XINCSQ + Y(PT) ** 2
         CRORVL(PT) = 1.0
         PVORVL(PT) = 0.0
         SUMOTH = SUMOTH + Y(PT)
100   CONTINUE
```

```
C
      SUMSQ(0) = SUMOTH / SORSQC
      CFORPY(0) = SUMSQ(0)
      XINCSQ = XINCSQ - SUMSQ(0) * SUMOTH
      SMOTH2(0) = XINCSQ / FLOAT( NUMPTS - 1 )
C
C     Transformation of the abcissa
      SCALE = 4.0 / ( XMAX - XMIN )
      XINC = 4.0 / FLOAT( NUMPTS )
      XZER = ( XMAX + XMIN ) * 0.5
C
C     Main computation loop
      LSTORD = ORDRQD - 1
      DO 300 ORD = 0, LSTORD, 1
         ORDP1 = ORD + 1
         DUMMY = 0.0
C
C        Compute the alpha term
         XVAL = -2.0
         DO 110 PT = START, LSTPT, 1
            DUMMY = DUMMY + XVAL * CORVL(PT) ** 2
            XVAL = XVAL + XINC
110      CONTINUE
         ALPHA(ORDP1) = DUMMY / SORSQC
         SORSQP = SORSQC
         SORSQC = 0.0
         SUMOTH = 0.0
C
C        Compute the beta term
         XVAL = -2.0
         DO 120 PT = START, LSTPT, 1
            DUMMY = BETA(ORD) * PVORVL(PT)
            PVORVL(PT) = CRORVL(PT)
            CRORVL(PT) = (XVAL-ALPHA(ORDP1)*CRORVL(PT)-DUMMY
            SORSQC = SORSQC + CRORVL(PT) ** 2
            SUMOTH = SUMOTH + Y(PT) * CRORVL(PT)
            XVAL = XVAL + XINC
120      CONTINUE
         BETA(ORDP1) = SORSQC / SORSQP
         SUMSQ(ORDP1) = SUMOTH  / SORSQC
         XINCSQ = XINCSQ - SUMSQ(ORDP1) * SUMOTH
         SMOTH2(ORDP1) = XINCSQ / FLOAT( NUMPTS - ORD - 1 )
C
         IF ( SWX ) GOTO 200
C
C           Higher power will not improve fit
            CFORPY(ORDP1) = 0.0
            GOTO 290
200      CONTINUE
            IF ( SMOTH2(ORDP1) .LT. SMOTH2(ORD) ) GOTO 210
C
C              Termination of loop when higher power will not
C              improve fit
               SWX = .FALSE.
               CFORPY(ORDP1) = 0.0
```

```
                GOTO 280
210           CONTINUE
C
C                Evaluate the polynomial coefficients
                 DO 220 DEG = 0, ORD, 1
                    DUMMY = CFORPV(DEG) * BETA(ORD)
                    CFORPV(DEG)  = CFORCR(DEG)
                    CFORCR(DEG) = CFORPV(DEG - 1) - ALPHA(ORDP1) *
     +                 CFORCR(DEG) - DUMMY
                    CFORPY(DEG) = CFORPY(DEG) + SUMSQ(ORDP1) *
     +                 CFORCR(DEG)
220              CONTINUE
                 CFORPY(ORDP1) = SUMSQ(ORDP1)
                 CFORCR(ORDP1) = 1.0
                 CFORPV(ORDP1) = 0.0
280           CONTINUE
290        CONTINUE
300     CONTINUE
C
C       Transformation of the polynomial
C       The coefficients of the polynomial in (SCALE*X + CONST) have
C       been evaluated.  This transformation evaluates the
C       coefficients of X.
        CONST = -XZER * SCALE
        CONPWR(0) = 1.0
        SCLFAC(0) = 1.0
        COEF(0) = CFORPY(0)
        DO 310 ORD = 1, ORDRQD
           SCLFAC(ORD) = 1.0
           CONPWR(ORD) = CONST * CONPWR(ORD - 1)
           COEF(0) = COEF(0) + CFORPY(ORD) * CONPWR(ORD)
310     CONTINUE
C
        DO 330 ORD = 1, ORDRQD, 1
           SCLFAC(0) = SCLFAC(0) * SCALE
           COEF(ORD) = CFORPY(ORD) * SCLFAC(0)
           POLYNO = 1
           DO 320 DEG = ORDP1, ORDRQD, 1
              SCLFAC(POLYNO)    =    SCALE    *    SCLFAC(POLYNO)    +
SCLFAC(POLYNO - 1)
              COEF(ORD) = COEF(ORD) + CFORPY(DEG) * SCLFAC(POLYNO)*
     +           CONPWR(POLYNO)
              POLYNO = POLYNO + 1
320        CONTINUE
330     CONTINUE
        RETURN
        END
```

End of Appendix

APPENDIX 3: HYBRID CURVE FITTING

The programs given in this appendix were used to fit the Pressure-Concentration curve. Two programs are given:

- Curve Fitting by Segmentation
- Curve Fitting by Transformation

Both these methods were written in Data General Fortran. The comments on this version of Fortran in Appendix 2 should be noted. The modularity of these programs could have been improved if it had been possible to automatically recompile all the segments quickly; however, with a Nova running on flexible diskettes, this is not the case.

Appendix 3.1: Segmentation

```
C
C     Written: 26/03/81, CUP
C
C     Test Driver for Fitting by Segmentation
C
      COMPILER DOUBLE PRECISION
      PARAMETER MXPTS = 100, MAXORD = 9, MXORD1 = 10
C
      INTEGER BLOCK,  DEG,    DEGFDM, DUMMY,  ERROR,  HALT,   I
      INTEGER NUMPTS, ORD,    OTFILE, PT,     STRT,   VOLTGE
      REAL    CF,     CHISQR, COEF,   DIFF,   F,      FIDFAC, FVAL
      REAL    MXDIFF, MX2,    RHALT,  RSTEP,  RSTRT,  SUMDIF
      REAL    XVAL,   Y,      YVAL
      REAL    CF(1:7,0:MXORD1), COEF(0:MXORD1), Y(MXPTS)
      REAL    VOLTGE(MXPTS)
C
      EXTERNAL CURFIT
      F( I ) = FLOAT( VOLTGE(I) )
C
C     Open Data File
      CALL OPEN( 20, 'CFRAW.DT', 1, ERROR )
      IF ( ERROR .EQ. 1 ) GOTO 100
         TYPE 'Error in opening CFRAW.DT - ', ERROR
         CALL EXIT
100   CONTINUE
C
C     Output File; 10 for TTO, 12 for LPT
      OTFILE = 10
      WRITE( OTFILE, 440 )
C
C     Main Loop
200   CONTINUE
C
C        Get Test Data
C        The following is DG's form of an unformatted read
         READ BINARY( 20, END = 500 )
     -      NUMPTS,                          ; Number of Points
     -      ( VOLTGE(I), I = 1, NUMPTS ),; The Data
     -      ( DUMMY, I = 1, 10, 1 )          ; A remark
C
C        Fit the curve in blocks
         RSTEP = FLOAT( NUMPTS ) / 8.0
         RSTRT = 1.0 - STEP
         RHALT = RSTEP
         DO 260 BLOCK = 1, 7, 1
C
C           Copy the block
            RSTRT = RSTRT + STEP
            RHALT = RHALT + STEP
            STRT  = INT( RSTRT )
            HALT  = MIN0( NUMPTS, INT( RHALT ) )
            DO 220 PT = STRT, HALT, 1
               Y(PT) = F( PT )
```

```
220         CONTINUE
C
            CALL CURFIT( FLOAT( STRT ), FLOAT( HALT ), Y,
     -         STRT, HALT, COEF, MXORD1 )
C
C           Store the coefficients
            DO 240 ORD = 0, MXORD1
               CF(BLOCK,ORD) = COEF(ORD)
240         CONTINUE
260      CONTINUE
C
C        Reproduce the curve
         DO 280 PT = 1, NUMPTS
            Y(PT) = 0.0
280      CONTINUE
C
         RSTRT = 1.0 - RSTEP
         RHALT = RSTEP
         DO 360 BLOCK = 1, 7, 1
C
C           Set up parameters
            RSTRT = RSTRT + RSTEP
            RHALT = RHALT + RSTEP
            STRT = INT( RSTRT )
            HALT = MIN0( NUMPTS, INT( RHALT ) )
            DO 300 ORD = 0, MXORD1
               COEF(ORD) = CF(BLOCK,ORD)
300         CONTINUE
C
C           Evaluate the points
            DO 340 PT = STRT, HALT, 1
               XVAL = FLOAT( PT )
               YVAL = COEF(MXORD1)
               DO 320 ORD = 1, MXORD1
                  YVAL = YVAL * XVAL + COEF(MXORD1 - ORD)
320            CONTINUE
               Y(PT) = Y(PT) + YVAL
340         CONTINUE
360      CONTINUE
C
C        Sort out the overlapping segments
         STRT = INT( 1.0 + RSTEP )
         HALT = INT( RHALT - RSTEP )
         DO 380 PT = STRT, HALT, 1
            Y(PT) = AINT(AMAX1(0.0,AMIN1(Y(PT)*0.5,255.0)))
380      CONTINUE
```

```
C
C       Evaluate Statistics
        MXDIFF = 0.0
        SUMDIF = 0.0
        CHISQR = 0.0
        DO 400 PT = 1, NUMPTS, 1
           XVAL = FLOAT( PT )
           YVAL = Y(PT)
           FVAL = F( PT )
           DIFF = ABS( YVAL - FVAL )
           SUMDIF = SUMDIF + DIFF
           MXDIFF = AMAX1( MXDIFF, DIFF )
           IF ( FVAL .GT. 0.0 )
     -         CHISQR = CHISQR + ( DIFF * DIFF ) / FVAL
400     CONTINUE
        DIFF = SUMDIF / FLOAT( NUMPTS + 1 )
        DEGFDM = NUMPTS - 1
C
C       The results
        WRITE( OTFILE, 420 ) NUMPTS, MXDIFF, DIFF, DEGFDM, CHISQR
420     FORMAT( 3X, I2, F15.5, F13.5, I7, F13.5 )
440     FORMAT( 1X, 'NUMPTS  MAX POS DIF  AVG POS DIF   DEG FDM',
     -      '   CHI SQUARE' / )
        GOTO 200
C
C     End of File
500   CONTINUE
      CALL EXIT
      END
```

Appendix 3.2: Results of Segmentation

The last column in the above table gives the approximate value of the chi-squared distribution at 99.5%. The odd results in the second row are due to the large fluctuations in Region 1 of the graph.

NUMPTS	MAX POS DIF	AVG POS DIF	DEG FDM	CHI SQUARE	Chi-Square
52	18.00000	2.68110	53	22.16677	28.7451
26	226.00000	33.79407	25	714.77909	10.5197
57	8.00000	2.63512	58	24.56095	32.5170
76	14.00000	2.76990	77	23.74200	47.2236
68	14.00000	2.40454	69	18.41089	40.9530

Appendix 3.3: Mathematical Transformation

```
C
C     Written: 26/03/81, CUP
C
C     Test Driver for Fitting by Mathematical Transformation
C
      COMPILER DOUBLE PRECISION
      PARAMETER MXPTS = 100, MAXORD = 9, MXORD1 = 10
C
      INTEGER DEG,    DEGFDM, DUMMY,  ERROR,  I
      INTEGER NUMPTS, ORD,    OTFILE, START,  VOLTGE
      REAL    CHISQR, COEF,   DIFF,   F,      FIDFAC, FVAL
      REAL    MXDIFF, MX2,    SUMDIF, XVAL,   Y,      YVAL
      DIMENSION COEF(0:MXORD1), Y(MXPTS), VOLTGE(MXPTS)
C
      EXTERNAL CURFIT
      F( I ) = FLOAT( VOLTGE(I) )
C
C     Open Data File
      CALL OPEN( 20, 'CFRAW.DT', 1, ERROR )
      IF ( ERROR .EQ. 1 ) GOTO 100
         TYPE 'Error in opening CFRAW.DT - ', ERROR
         CALL EXIT
100   CONTINUE
C
C     Output File; 10 for TTO, 12 for LPT
      OTFILE = 10
      WRITE( OTFILE, 440 )
      FIDFAC = 2.0 ; The fiddle factor
C
C     Main Loop
200   CONTINUE
C
C        Get Test Data.
C        The following is DG's form of an unformatted read
statement
         READ BINARY( 20, END = 500 )
     -      NUMPTS,                        ; Number of Points
     -      ( VOLTGE(I), I = 1, NUMPTS ),; The Data
     -      ( DUMMY, I = 1, 10, 1 )        ; A remark
C
C        Find the starting position
         DO 205 START = 1, NUMPTS
            IF ( VOLTGE(START) .GT. 10 ) GOTO 210
205      CONTINUE
C
C        Hopefully there will be an exit from the
C        loop before numpts is reached
210      CONTINUE
         START = MAX0( START - 3, 1 )
```

```
C
C       Transform Data
        MX2 = FIDFAC * FLOAT( NUMPTS - START + 1 )
        DO 240 PT = START, NUMPTS, 1
           XVAL = FLOAT( I )
           FVAL = F( I )
           Y(I) = FVAL * XVAL / ( MX2 - XVAL )
240     CONTINUE
C
C       Fit the test data
        CALL CURFIT( FLOAT( START ), FLOAT( NUMPTS ), Y,
     -          START, NUMPTS, COEF, MXORD1 )
C
C       Evaluate Statistics
        MXDIFF = 0.0
        SUMDIF = 0.0
        CHISQR = 0.0
        DO 300 PT = START, NUMPTS, 1
           XVAL = FLOAT( PT )
           YVAL = COEF(MXORD)
C
C          Evaluate the Polynomial
           DO 260 ORD = 1, MXORD, 1
              YVAL = YVAL * XVAL + COEF(MXORD - ORD)
260        CONTINUE
C
C          Reverse Transformation
           YVAL = AINT( YVAL * ( MX2 - XVAL ) / XVAL )
C
C          Correct result if it is out of range
           IF ( YVAL .LT. 0.0 ) YVAL = 0.0
           IF ( YVAL .GT. 255.0 ) YVAL = 255.0
           Y(PT) = YVAL
C
C          Evaluate the differences
           FVAL = F( PT )
           DIFF = ABS( YVAL - FVAL )
           SUMDIF = SUMDIF + DIFF )
           IF ( FVAL .GT. 0.0 ) CHISQR = CHISQR + ( DIFF * DIFF )
/ FVAL
300     CONTINUE
C
C       Evaluate the differences in Region 1
        FVAL = F( START )
        DO 400 PT = 1, START
           XVAL = FLOAT( I )
           Y(I) = YVAL
           FVAL = F( I )
           DIFF = ABS( YVAL - FVAL )
           SUMDIF = SUMDIF + DIFF )
           IF ( FVAL .GT. 0.0 ) CHISQR = CHISQR + ( DIFF * DIFF )
/ FVAL
300     CONTINUE
        DIFF = SUMDIF / FLOAT( NUMPTS + 1 )
        DEGFDM = NUMPTS - 1
```

```
C
C       The results
        WRITE( OTFILE, 420 ) NUMPTS, MXDIFF, DIFF, DEGFDM, CHISQR
420     FORMAT( 3X, I2, F15.5, F13.5, I7, F13.5 )
440     FORMAT( 1X, 'NUMPTS  MAX POS DIF  AVG POS DIF   DEG FDM',
     -      '   CHI SQUARE' / )
        GOTO 200
C
C     End of File
500   CONTINUE
      CALL CLOSE( 20 )
      CALL EXIT
      END
```

Appendix 3.4: Results of Curve Fitting by Transformation

The last column in the above table gives the approximate value of the chi-squared distribution at 99.5%. The author cannot find any reasons for the large differences in the goodness of fit for the first three sets of data. It could be that the transformation works better for a larger number of points (eg $\frac{3}{8}$65) or it could be just pure luck that there was a good curve fit in the fourth and fifth sets of data.

NUMPTS	MAX POS DIF	AVG POS DIF	DEG FDM	CHI SQUARE	Chi-Square
52	15.00000	6.84906	53	232.25162	28.7451
26	31.00000	11.96296	25	54.84466	10.5197
57	31.00000	7.20690	58	174.95897	32.5170
76	18.00000	4.58442	77	42.17616	47.2236
68	12.00000	4.59420	69	40.68316	40.9530

Appendix 3.5: Test Data

The test data is in its formatted form for readability; it would normally be in the internal unformatted form. It has been scaled up to the range 0 to 255 which is the form in which the PET would send it.

```
52
   4.   0.   1.   1.   1.   4.   0.   4.   8.  14.
  33.  42.  68.  93. 106. 128. 140. 153. 164. 169.
 174. 179. 183. 183. 189. 194. 196. 200. 203. 204.
 205. 209. 213. 211. 217. 215. 217. 226. 221. 226.
 230. 230. 229. 232. 234. 234. 230. 238. 238. 238.
 243. 238.
Kink in Region 3 & 4
26
  10.   4.   8.   8.  10.   8.   8.  21.  29.  59.
  76.  93. 123. 136. 159. 204. 217. 221. 226. 226.
 226. 230. 231. 234. 234. 238.
Short batch of data
57
   5.   1.   1.   4.   0.   2.   0.   1.   5.   4.
   0.   5.   0.   0.   3.   4.  11.  11.  11.  22.
  29.  35.  51.  73.  88. 106. 135. 126. 138. 142.
 157. 159. 170. 182. 182. 184. 190. 193. 193. 204.
 204. 210. 206. 206. 207. 219. 216. 216. 218. 219.
 225. 219. 226. 219. 223. 225. 227.
Wavy 3rd & 4th region
76
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
   0.   0.   0.   0.   7.  10.  24.  36.  54.  63.
  76. 102. 115. 117. 131. 138. 146. 160. 157. 168.
 168. 168. 177. 179. 175. 175. 181. 175. 188. 182.
 186. 186. 188. 186. 196. 191. 193. 194. 196. 196.
 201. 197. 203. 199. 199. 204. 206. 204. 203. 210.
 207. 214. 210. 207. 206. 214. 212. 212. 212. 212.
 219. 219. 215. 219. 223. 223.
Long Region 1
68
   0.   0.   3.   0.   3.   4.   8.  11.  17.  27.
  39.  51.  64.  83.  89. 115. 121. 128. 128. 145.
 157. 161. 161. 164. 168. 168. 172. 174. 174. 175.
 181. 179. 180. 183. 184. 187. 183. 187. 188. 189.
 189. 191. 191. 198. 196. 192. 196. 198. 195. 191.
 194. 192. 198. 195. 191. 194. 196. 198. 200. 204.
 206. 204. 202. 200. 199. 198. 197.
Long Region 3
```

End of Appendix

APPENDIX 4: DATA COMMUNICATIONS

Appendix 4.1: Procedure for copying COMMLINK

The program given here [COMMPACK] can be used for both copying and loading COMMLINK [Taylor Wilson [1980]]. In order to copy COMMLINK,

- Form a binary version of COMMPACK
- Load COMMLINK from the Master Disk
- Load COMMPACK
- Type 'SYS 1422'
- Save COMMPACK

COMMPACK now contains COMMLINK concatenated below it. To load COMMLINK into memory,

- Load COMMPACK
- Type 'RUN'
- Type in the password [this will not be disclosed: it is left as a programming exercise to the reader; after all, the listing is given]

After loading, any programs which use COMMLINK may be executed.

Appendix 4.2: COMMLINK Copier

```
;      put "@0:commpack.src"
;      put "@1:commpack.src"
;
;      written: 08/07/81, cup
;      updated: 08/07/81, cup
;
;      purpose: copy/load commlink
;
;-------------------------------
       *=  1025
input  =   133
rem    =   143
clrscr =   147            ; clear screen char
print  =   153
clr    =   156
sys    =   158
plus   =   170
eq     =   178
left   =   200
;---------------------------------------
; the basic loader
       .wor link1,10000
       .byt rem,'==communications link',0
link1  = *
       .wor link2,10020
       .byt rem,'  written: 08/07/81, cup',0
link2  = *
       .wor link3,10040
       .byt rem,'  updated: 08/07/81, cup',0
link3  = *
       .wor link4,10100
       .byt clr,':'
       .byt input,'"password";s$:'
       .byt print,'"',clrscr,0
link4  = *
       .wor link5,10120
       .byt 's$',eq,left,'(s$+"              ",12)',0
link5  = *
       .wor link6,10140
       .byt sys,'1320',0
link6  = *
       .wor link7,63998
       .byt rem,'  open1,8,15,"s0:communicator":'
       .byt 'save"0:communicator",8:'
       .byt 'verify"*",8',0
link7  = *
       .wor link8,63999
       .byt rem,'  open1,8,15,"s1:communicator":'
       .byt 'save"1:communicator",8:'
       .byt 'verify"*",8',0
link8  = *
       .wor 0,0
;---------------------------------------
```

```
;  declarations
;  ------------
       *=  1320
strptr =   17              ; string pointer
varptr =   42              ; start of variables
newadr =   44              ; new address
topmem =   53              ; top of memory
chrget =   112             ; system's lexical analyzer
kbdix  =   158             ; keyboard index
oldptr =   238
newptr =   240
kbdbuf =   623             ; keyboard buffer
comstr =   $7200           ; start of commlink
ready  =   $b3ff           ; warm start
cold   =   $fffc           ; cold start
;-------------------------------------
; check the password
.ski 2
       ldy £3              ; pointer to password
       lda (varptr),y
       sta strptr
       iny
       lda (varptr),y
       sta strptr+1
       ldy £$ff            ; check password
       tya
       sta temp
incy   = *
       iny
       cpy £12             ; no of chars in password
       beq load
       lda (varptr),y
       eor £$ff
       eor temp
       sta temp
       tya
       adc  temp
       cmp passwd,y
       beq incy
passwd .dby $36fb,$38f5,$3df9,$45a2,$8663,$c724
;-------------------------------------
load   = *
       ldx £02             ; overwrite chrget
newscn = *
       lda newchr,x
       sta chrget,x
       dex
       bpl newscn
       lda £⅜comstr        ; set up load address
       ldx £⅛comstr
       sta newadr
       stx newadr+1
       sta topmem
       stx topmem+1
       lda £⅛comsto        ; storage position
```

```
        ldx £⅜comsto
        sta oldptr
        stx oldptr+1
        jsr copy
        ldx £6              ; clobber keyboard buffer with 'new'
        stx kbdix
cpynew = *
        lda newbyt-1,x
        sta kbdbuf-1,x
        dex
        bne cpynew
        jmp ready
;-----------------------------
; copy commlink from $7200 to bottom of program
; and reset the end of program.  this routine
; should be entered when copying commlink from
; memory.
revers = *
        lda £⅛comsto        ; storage position
        ldx £⅜comsto
        sta newadr
        stx newadr+1
        lda £⅛comstr; new location
        ldx £⅜comstr
        sta oldptr
        stx oldptr+1
        jsr copy
        lda £⅛endprg        ; mark new end of program
        ldx £⅜endprg
        sta varptr
        stx varptr+1
        rts
;-----------------------------
; copy 14 blocks from [oldptr] to [newptr]
copy    = *
        ldx £14             ; number of blocks to be copied
        ldy £0
copy1   = *
        lda (oldptr),y
        sta (newptr),y
        iny
        bne copy1
        inc oldptr+1        ; next block
        inc newptr+1
        dex
        bne copy1
        rts
;-----------------------------
temp    *=*+1               ; temp location
newchr jmp $7601            ; new top of chrget
newbyt .byt 'new',$0d,$07,clrscr
comsto *=*+$e00             ; copy of commlink starts here
endprg = *
.end
```

Appendix 4.3: Data Transmitter

This appendix and appendices 4.4 and 4.5 give a simplified version of the data transmitter. It does not transmit floating point numbers. The main reason for this is that the receiving end (ie the Nova) did not have any form of free formatted read. To write a parser for the various formats of floating point numbers which the PET could generate is quite a task, especially in Fortran and would defeat the purpose of the exercise which was to check that the hardware and software interfaces were working.

```
10000  REM==TEST TRANSFER PROGRAM
10020  REM  WRITTEN: 08/07/81, CUP
10040  REM  UPDATED: 08/07/81, CUP
10100  CMD I
10120  CMD B,1024*10
10140  X$="":
       CR$=CHR$(13)
10160  CMD F,C=Y,D=22,DI=N,EX=Y,SE=13,E=3,S=-1,T=5,R=255
10180  REM  EXECUTE PROGRAM ON DG
       X$="receive"+CHR$(13):
10200  CMD S,X$
10220  INPUT "[rvs.on]Pet Filename [rvs.off] : ";FL$
10240  INPUT "[rvs.on]Drive Number [rvs.off] : ";DR
10260  DOPEN £2,(FL$),D(DR)
10270  INPUT £2,CT$,NC,SM,SS:
       X$=STR$(NC*SM)+CR$:
       CMD S,X$
10280  REM--SKIP OVER NAMES & UNITS
       FOR I=1 TO NC:
          INPUT £1,NM$,UN$:
       NEXT I:
10300  FOR I=1 TO NC*SM STEP 12

10320     REM--PAD X$ TO 72 CHARS
          X$="123456789":
          FOR J=1 TO 3:
             X$=X$+X$:
          NEXT J:
10340     SYS 634:
          CMD S,X$:
          IF ST=0 THEN
          NEXT I
10360  DCLOSE £2:
       X$="9999"+CR$:
       CMD S,X$
63999  REM SCRATCH"xfer to dg",D0:DSAVE"xfer to dg",D0:VERIFY"*",8
```

Appendix 4.4: Format Converter

```
.opt lis,err,nog
;       put "@0:format.src"
;       put "@1:format.src"
;
;       written: 08/07/81, cup
;       updated: 08/07/81, cup
;
;       purpose: convert format of number to octal
;
;-------------------------------------
varbeg = 42             ; start of variables
strptr = 23             ; pointer to beginning of output string
posn   = 0              ; current output position
status = 150            ; ieee status
data   = 17             ; temporary data storage bytes
setget = $ffc6          ; set input channel
get    = $ffe4          ; get a byte
reset  = $ffcc          ; reset to default i/o
       * = 634
       ldy £2           ; assume first varb is a string
       lda (varbeg),y
       sta strptr+1
       dey
       lda (varbeg),y
       sta strptr
       dey               ; y is now zero
       lda £' '          ; clear buffer
clear  = *
       sta (strptr),y
       iny
       cpy £72
       bne clear
       lda £0           ; position in string
       sta posn
nexnum = *
       jsr getbyt       ; get the data
       ldx status       ; end of file?
       beq eof
       sta data         ; hi byte
       jsr getbyt
       sta data+1
       ldx £4           ; convert to octal
       ldy posn
       iny
       iny
nexdig = *
       lda £0           ; get the ms digit into a
       asl data+1
       rol data
       rol a
       asl data+1
       rol data
       rol a
```

```
        asl data+1
        rol data
        rol a
        ora £$30        ; add ascii bias
        iny
        sta (strptr),y
        dex
        bne nexdig
        sty posn
        cpy £72
        bne nexnum
eof     = *
        rts
;-----------------------------------
; get a byte from channel 2
;
getbyt = *
        ldx £2
        jsr setget
        jsr get
        pha
        jsr reset
        pla
        rts
        .end
```

Appendix 4.5: Data Receiver

```
C
C     Written: June 1981, CUP
C     Data Receiver for concurrent running with Data Transmitter
C     on PET
C
      INTEGER LOGDAT(12)
C
C     Open output file
      CALL OPEN( 20, 'TESTDATA', 3, IERROR )
      IF ( IERROR .NE. 1 ) GOTO 100
         TYPE 'Error in opening test data file', IERROR
         CALL EXIT
100   CONTINUE
C
C     Main Loop
110   CONTINUE
C
C        Use DG's form of free formatted read from the terminal
         READ( 11,120 ) NUMPTS
120      FORMAT( I3 )
         WRITE BINARY( 20 ) NUMPTS
C
C        Get the data
         DO 170 LSET = 1, NUMPTS, 12
            READ( 11, 130 ) ( LOGDAT(I), I = 1, 12, 1 )
130         FORMAT( 12 I6 )
C
C           Convert to binary
            DO 150 I = 1, 12, 1
               NUMBER = LOGDAT(I)
               IF ( NUMBER .GT. 7777 ) GOTO 160
               NEWVAL = 0
               IOCTAL = 1
               DO 140 LPOSN = 1, 4, 1
                  NEWVAL = NEWVAL + MOD( NUMBER, 10 ) * IOCTAL
                  NUMBER = NUMBER / 10
                  IOCTAL = IOCTAL * 8
140            CONTINUE
               LOGDAT(I) = NUMBER
150         CONTINUE
C
C           Assume I is 13 on exit from loop
160         CONTINUE
            I = I - 1
C
C           Write to file
            WRITE BINARY( 20 ) ( LOGDAT(J), J = 1, 12, 1 )
170      CONTINUE
C
C        End of data or end of file
```

```
            IF ( NUMBER .EQ. 8888 ) GOTO 110
C
C       End of file
        CALL FCLOS( 20 )
        CALL EXIT
        END
```

End of Appendix

APPENDIX 5: THE B-TREE ALGORITHM

The program given here shows how variable length keys may be stored in a B-tree. This was a suggestion due to Martin, quoted in Knuth [1969b]. It was developed on the CBM 8032 in BASIC.

The program generates pseudo-random variable length keys and inserts them into the B-tree. The tree is printed after every 20 insertions in one of two selectable formats:

- In what is known in jargonese as a suffix walk. This displays the keys in sorted order, as they are stored in the conceptual tree.
- In sequential pages, that is, the order in which they are stored on disk.

Appendix 5.1: B-tree program

```
10000  REM==B-TREE MAINTENANCE
10020  REM  WRITTEN: 15/01/81, CUP
10040  REM  UPDATED: 18/01/81, CUP
10060  REM  PURPOSE: STORAGE OF VARIABLE LENGTH KEYS IN A B-TREE

10100  REM==DRIVER
10120  REM--INITIALIZE
       GOSUB 50000:

10140  REM--INSERT 1000 KEYS; START AT 1000 TO GET LEADING ZEROS
10160  FOR K=1000 TO 1999 STEP 20:
          FOR I=K TO K+19

10180        REM--FORM A STRING OF L CHARS
             S$=RIGHT$(STR$(I),3):
             L=INT(RND(1)*5)+2:
10200        FOR J=1 TO L:
                S$=CHR$(INT(RND(0)*26)+A)+S$:
             NEXT J

10220        REM--INSERT S$ IN B-TREE
             PRINT I-1000,S$:
             GOSUB 30000:
          NEXT I:

10240     REM--WALK THE TREE
          GOSUB 40000:
          GOSUB 41000:
       NEXT K:

10280  REM--TERMINATE
       GOSUB 40000:
       GOSUB 41000:
       GOSUB 29000:
10300  IF NOT UM THEN
          DCLOSE £2
10320  END

20000  REM==GET PAGE PR INTO BUFFER PP
20001  REM  GLOBAL: AL(), LP, PG(), PG$()
20002  REM  LOCAL : P
20003  REM  OWN   : LR(), TM
20004  REM  PARAM : PP, PR

20040  REM--IS IT ALREADY IN TABLE?
       FOR PP=1 TO MP:
          IF PR=PG(PP) GOTO 20500:
20060  NEXT PP

20080  REM--FLAG PAGE FAULT
       POKE 32847,42:
```

```
20100  REM--LOOK FOR LEAST RECENTLY USED PAGE
       PP=1:
20120  FOR P=2 TO MP:
          IF LR(P) lt LR(PP) THEN
             PP=P
20140  NEXT P

20200  REM--WRITE IF ALTERED
       IF AL(PP) THEN
          GOSUB 23000:

20300  REM--READ IF IT EXISTS
       IF PR le LP THEN
          GOSUB 24000:
20400  PG(PP)=PR:
       AL(PP)=UU:
       POKE 34895,32
20420  TM=TM+1:
       LR(PP)=TM:
       RETURN

20500  REM--UPDATE LRU COUNTER
20520  TM=TM+1:
       LR(PP)=TM:
       RETURN

21000  REM==GET A NEW PAGE
21001  REM  GLOBAL: LP, PR
21002  REM  LOCAL : T
21004  REM  PARAM : NP, PN
21020  T=PP:
       PR=LP+1:
       GOSUB 20000:
       LP=PR:
       NP=LP:
       PG$(PP)=Z$:
       PN=PP:
       PP=T:
       RETURN

23000  REM==WRITE A RECORD
23001  REM  GLOBAL: AL(), BT$(), ER$, PG(), PG$(), UM
23002  REM  LOCAL : RC$
23004  REM  PARAM : PP
23020  IF UM THEN
          BT$(PG(PP))=PG$(PP):
          RETURN
23040  RECORD £2,(PG(PP)):
       RC$=PG$(PP)+ER$:
       PRINT £2,RC$;:
       RETURN
```

```
24000  REM==READ A RECORD
24001  REM  GLOBAL: BT$(), ER$, PG(), PG$(), UM, Z$
24002  REM  LOCAL : C$
24004  REM  PARAM : PP, PR
24020  IF UM THEN
          PG$(PP)=BT$(PR):
          RETURN
24040  PG$(PP)="":
       RECORD £2,(PR)
24060  GET £2,C$:
       IF C$="" THEN
          C$=Z$
24080  IF C$ ne CR$ THEN
          PG$(PP)=PG$(PP)+C$:
          GOTO 24060
24100  RETURN

29000  REM==DUMP ALL PAGES
29020  FOR PP=1 TO MP:
          IF AL(PP) THEN
             GOSUB 23000
29060  NEXT PP
29080  RETURN

30000  REM==SEARCH FOR S$

30020  REM--INITIALIZE
       SP=-1:
       CN=RT:
       PN$=Z$:
       IS$=CHR$(LEN(S$))+S$:
       IF RT=0 GOTO 31600:

30100  REM--GET PAGE
       PR=CN:
       GOSUB 20000:
       CP$=PG$(PP):

30120  REM--PERFORM SEQUENTIAL SEARCH WITHIN PAGE
       CP=3:
       FOR S=1 TO FNMA(1):

30140     REM--EXTRACT STRING
          L=FNMA(CP):
          CS$=MID$(CP$,CP+1,L):
30160     IF CS$=S$ THEN
             PRINT "found":
             RETURN
30180     IF CS$ lt S$ THEN
             CP=CP+L+2:
          NEXT S
```

```
30200  REM--GO DOWN MIDDLE BRANCH
       NN=FNMA(CP-1):
30220  IF NN ne 0 THEN
          SP=SP+1:
          S(SP,PA)=CN:
          CN=NN:
          S(SP,PO)=CP:
          GOTO 30100

31000  REM==INSERT S$

31040  REM--ASSUME PP, CP ARE CORRECT
31200  CP$=PG$(PP):
       AL(PP)=CH:
       PG$(PP)=CHR$(FNMA(1)+1)+MID$(CP$,2,1)

31220  REM--COPY FRONT
       IF CP gt 3 THEN
          PG$(PP)=PG$(PP)+MID$(CP$,3,CP-3):

31240  REM--INSERT
       PG$(PP)=PG$(PP)+IS$+PN$:

31260  REM--COPY BACK
       IF CP le LEN(CP$) THEN
          PG$(PP)=PG$(PP)+MID$(CP$,CP):

31280  REM--OVERFLOW?
       IF LEN(PG$(PP)) le PZ THEN
          RETURN:

31300  REM--FIND THE SPLITTING POINT
       CP=3:
       CP$=PG$(PP):
       S=0:
31400  S=S+1:
       PV=CP:
       CP=CP+2+FNMA(CP):
       IF CP lt P2 GOTO 31400

31500  REM--EXTRACT THE STRING TO BE PROMOTED
       IS$=MID$(CP$,PV,FNMA(PV)+1):

31520  REM--RE-DO OLD NODE
       AL(PP)=CH:
       PG$(PP)=CHR$(S-1)+MID$(CP$,2,PV-2):

31540  REM--INSERT NEW NODE
       GOSUB 21000:
31560  AL(PN)=CH:
       PG$(PN)=CHR$(FNMA(1)-S)+MID$(CP$,CP-1):
       PN$=CHR$(NP)
```

```
31580  IF SP ge 0 THEN
          CN=S(SP,PA):
          CP=S(SP,PO):
          SP=SP-1:
          PR=CN:
          GOSUB 20000:
          GOTO 31200

31600  REM--TREE GROWING IN HEIGHT: NEW ROOT
       GOSUB 21000:
       AL(PN)=CH:
31620  PG$(PN)=CHR$(1)+CHR$(RT)+IS$+PN$:
       RT=NP:
       RETURN

40000  REM==WALK THE TREE
40020  INPUT "Walk Tree";YN$:
       IF YN$="n" THEN
          RETURN

40040  REM--INITIALIZE
       CN=RT:
       SP=-1:
       OPEN 222,DV:

40100  REM--BEGINNING OF A NEW PAGE
       CP=3:

40200  REM--CHECK THE BRANCH
       PR=CN:
       GOSUB 20000:
       NN=FNPG(CP-1):
40220  IF NN gt 0 THEN
          SP=SP+1:
          S(SP,PA)=CN:
          S(SP,PO)=CP:
          CN=NN:
          GOTO 40100

40300  REM--END OF PAGE?
       IF CP gt LEN(PG$(PP)) GOTO 40400:
40320  IF NN ne 0 OR CP=3 THEN
          PRINT £222:
          PRINT £222,LEFT$(B$,(SP+1)*5);
40340  L=FNPG(CP):
       PRINT £222,MID$(PG$(PP),CP+1,L);" ";

40360  REM--MOVE TO NEXT STRING
       CP=CP+L+2:
       GOTO 40200:

40400  REM--END OF NODE: GET PARENT
```

```
40420  REM--END OF WALK?
       IF SP lt 0 THEN
          PRINT £222:
          CLOSE 222:
          RETURN:
40440  CN=S(SP,PA):
       CP=S(SP,PO):
       SP=SP-1:
       PR=CN:
       GOSUB 20000:
       NN=1:
       GOTO 40300

41000  REM==SEQUENTIAL DUMP OF PAGES
41020  INPUT "Sequential Dump";YN$:
       IF YN$="n" THEN
          RETURN
41030  OPEN 222,DV:
       PRINT £222,"sequential dump of pages":
       FOR PR=BE TO LP
41040     GOSUB 20000:
          CP$=PG$(PP):
          CP=3:
          PRINT £222,"page";PR;FNMA(2);
41060     FOR S=1 TO FNMA(1):
             L=FNMA(CP):
             PRINT £222,MID$(CP$,CP+1,L);
41080        CP=CP+L+2:
             PRINT £222,FNMA(CP-1);" ";:
          NEXT S
41100     PRINT £222:
       NEXT PR:
       CLOSE 222:
       RETURN

50000  REM==INITIALIZE
50010  A=ASC("a")
50020  B$=" ":
       BE=1
50030  C$="":
       CH=-1:
       CP=0:
       CP$=""
50040  DV=4
50050  ER$=CHR$(255)
50090  I=0:
       IS$=""
50100  J=0
50120  L=0:
       LP=BE-1
50130  MP=6:
       DEF FNMA(I)=ASC(MID$(CP$,I,1))
50140  NN=0:
       NP=0
50160  P=0:
```

```
        PA=0:
        PN=0:
        PN$="":
        PO=1:
        PP=0:
        PR=0:
        PV=0:
        PZ=80:
        P2=INT(PZ/2)
50162   DEF FNPG(I)=ASC(MID$(PG$(PP),I,1))
50180   RC$="":
        RT=0
50190   S=0:
        S$="":
        SP=0
50200   T=0:
        TM=-65535
50210   UM=0:
        UU=0
50260   Z$=CHR$(0)

51000   REM--ARRAYS
51010   DIM AL(MP)
51020   DIM BT$(255)
51120   DIM LR(MP)
51160   DIM PG(MP),PG$(MP)

52000   REM--INITIALIZATION
52020   FOR I=1 TO MP:
           AL(I)=UU:
           LR(I)=TM:
           PG(I)=0:
           PG$(I)="":
        NEXT I

52080   REM--IF THE REPLY TO THE FOLLOWING IS 'Y' THEN THE B-TREE
52100   REM  WILL BE STORED ON DISK OTHERWISE IT WILL BE STORED IN
        'CORE'
52120   INPUT "[2scr.home][scr.clr][cur.dwn]Use Disk";C$:
        UM=C$="n"
52140   IF NOT UM THEN
           DCLOSE :
           SCRATCH "btree.dt",D1:
           DOPEN £2,"btree.dt",L(PZ+1),D1
52200   POKE 224,1

52220   REM--FORM A STRING OF 128 BLANKS
        FOR I=1 TO 6:
           B$=B$+B$:
        NEXT I:
59999   RETURN
63998   REM     SCRATCH"btree.ds",D0:DSAVE"btree.ds",D0:VERIFY"*",8
63999   REM     SCRATCH"btree.ds",D1:DSAVE"btree.ds",D1:VERIFY"*",8
```

Appendix 5.2: Tree-Walk

WALKING THE TREE WITH 100 KEYS

AFMHTZ011 AO048 AP051 AQSY025 AVIL086 AYK034 BA027
BEWUC028
BLAL046 BLIKY088 BRJ078 BXMG015
CGFNVF042
CGGKWB077 CMYZ037 CSX013 CWBM084 DGNVP080 DSC057 EEZA022
EHFHL001
EKLE039 EOZP065 EQM085 EUW003 FHOPA052 FM089 GGGOQR063
GGHOQB033
GWT041 GXOJ083 HNOH097 HZVTM099
ICTUWG017
IR075 JEELTO020 JFQL047 JPPK064 JUHK053
KA000
KCUHIW023 KI092 KT026 KU091 LDELAL060
LGRM058
LHDR076 LK009 LPC038 MFIKN049 MHL056 MHTE055 MHTO073
MV019
NCFI094 NCPV082 NJVD018 NL079 NS072 OD029 OEQR066
OFFXDE014
OPA043 OV068 PDKWF061 PFK040 PNYQOI004 QRY031
QZ006
REPNVE010 RRHPWH071 SCE054 SDXI090 SG098 SKBM050
SQYSI032
TALEYJ070 TBTVW005 TDCUGH087 TG074 TOK096
UILXI016
USFZZ036 VDLIL035 VF062 VNPU059 VV067
WKLZUP012
WV069 XMQJLA002 XXIFHV030 XXHO081 XXSTRM024
YNDVIM008
YWH044 YWY021 ZACY007 ZENGSN093 ZIFBM095 ZTGG045

Appendix 5.3: Sequential Dump of Pages

The numbers in between the tags are pointers to the pages containing the branch nodes. The root node is on page 13.

SEQUENTIAL DUMP OF PAGES

PAGE 1 0 AFMHTZ011 0 AO048 0 AP051 0 AQSY025 0 AVIL086 0 AYK034 0 BA027 0

PAGE 2 0 REPNVE010 0 RRHPWH071 0 SCE054 0 SDXI090 0 SG098 0 SKBM050 0

PAGE 3 1 BEWUC028 7 CGFNVF042 17 EHFHL001 4 GGHOQB033 9 ICTUWG017 18

PAGE 4 0 EKLE039 0 EOZP065 0 EQM085 0 EUW003 0 FHOPA052 0 FM089 0 GGCOQR063 0

PAGE 5 0 KCUHIW023 0 KI092 0 KT026 0 KU091 0 LDELAL060 0

PAGE 6 0 WV069 0 XMQJLA002 0 XXIFHV030 0 XXHO081 0 XXSTRM024 0

PAGE 7 0 BLAL046 0 BLIKY088 0 BRJ078 0 BXMG015 0

PAGE 8 0 NCFI094 0 NCPV082 0 NJVD018 0 NL079 0 NS072 0 OD029 0 OEQR066 0 OFFXDE014 0

PAGE 9 0 GWT041 0 GXOJ083 0 HNOH097 0 HZVIM099 0

PAGE 10 0 TALEYJ070 0 TBTVW005 0 TDCUGH087 0 TG074 0 TOK096 0

PAGE 11 0 LHDR076 0 LK009 0 LPC038 0 MFIKN049 0 MHL056 0 MHTE055 0 MHTO073 0

PAGE 12 5 LGBM058 11 MV019 8 OFFXDE014 14 QZ006 2 SQYSI032 10 UILXI016 16 WKLZUP012 6 YNDVIM008 15

PAGE 13 3 KA000 12

PAGE 14 0 OPA043 0 OV068 0 PDKWF061 0 PFK040 0 PNYQOI004 0 QRY031 0

PAGE 15 0 YWH044 0 YWY021 0 ZACY007 0 ZENGSN093 0 ZIFBM095 0 ZTGG045 0

PAGE 16 0 USFZZ036 0 VDLIL035 0 VF062 0 VNPU059 0 VV067 0

PAGE 17 0 CGGKWB077 0 CMYZ037 0 CSX013 0 CWBM084 0 DGNVP080 0 DSC057 0 EEZA022 0

PAGE 18 0 IR075 0 JEELTO020 0 JFQL047 0 JPPK064 0 JUHK053 0

End of Appendix

APPENDIX 6: SIMPLE IMPLEMENTATION OF THE QUERY LANGUAGE

The program in Appendix 6.1 uses an Extended Basic system for the PET currently being developed by R. Narayanan and U.P. Cheah. The functions used are defined as follows:

- INPUT — This is similar to a normal INPUT function except that the line, column and various other parameters may be specified. Although it will accept seven different formats, only one is used in this program; that is function 0, the string input procedure.
- POKE — Display a piece of text either normally or in reverse field on a specified line and column.
- RESTORE — Reset the stack and return to the position of the last SYSTM[SAVE] command. This function is extremely useful for aborting after errors without going through the complicated procedure of unwinding the stack.
- SAVE — Note the return position for the next SYSTM[RESTORE] command.

This subroutine library should be available for general use by June 1982.

Appendix 6.1: Query Program

```
10000  REM==QUERIES
10020  REM  WRITTEN: 19/01/81, CUP
10040  REM  UPDATED: 01/03/81, CUP

10100  REM==DRIVER
10120  IF PEEK(53) ne 120 THEN
          POKE 53,120:
          CLR:
          DLOAD "system",D0

10140  REM--INITIALIZE
       GOSUB 50000:

10200  REM--GET QUERY
       GOSUB 40000:
10220  IF OP(1) ne 10 THEN
          GOSUB 42000:
          GOSUB 44000:
          GOTO 10200

10240  REM--TERMINATE
       END:

20000  REM==COMPUTE QUERY
20020  FOR LQ=NL TO 1 STEP -1
20040     IF OP(LQ) lt 7 THEN
             RS=RC(F(LQ)):
             LS=TV(LQ):
             GOTO 20200

20060     REM--LOGICAL OPERATORS
          RS=0:
20080     FOR LS=F(LQ) TO TV(LQ)
20100        REM  SUBTRACT SINCE TRUE IS -1
             RS=RS-R(LI(LS)):
20120     NEXT LS
20200     ON OP(LQ) GOTO 20250, 20300, 20350, 20400, 20450, 20500,
          20550, 20600,20650

20250     REM--less than
          RS=RS lt LS:
          GOTO 20700:

20300     REM--equal to
          RS=RS=LS:
          GOTO 20700:

20350     REM--greater than
          RS=RS gt LS:
          GOTO 20700:
```

```
20400     REM--not less
          RS=RS ge LS:
          GOTO 20700:

20450     REM--not equal
          RS=RS ne LS:
          GOTO 20700:

20500     REM--not greater
          RS=RS le LS:
          GOTO 20700:

20550     REM--AND
          RS=RS=(TV(LQ)-F(LQ)+1):
          GOTO 20700:

20600     REM--IOR
          RS=RS gt 0:
          GOTO 20700:

20650     REM--XOR
          RS=RS=1:
20700     R(LQ)=RS:
       NEXT LQ:
       RETURN

21000  REM==READ A RECORD
21020  FOR F=1 TO NF:
          INPUT £2,RC(F):
       NEXT F:
       RETURN

22000  REM==PRINT A RECORD
22020  DL$="":
       FOR F=1 TO NF:
          DL$=DL$+RIGHT$(B$+STR$(RC(F)),FW):
       NEXT F

23000  REM==DISPLAY A LINE
23020  IF DL ne PM GOTO 23060
23040  DL=0:
       SYS TM[POKE,0,23,1,"Hit space bar to continue"]
23050  GET C$:
       IF C$ ne " " GOTO 23050
23060  IF DL=0 THEN
          PRINT £3,HE$:
          PRINT "[scr.clr]";HE$
23080  DL=DL+1:
       PRINT £3,DL$:
       PRINT DL$:
       RETURN
```

```
33000  REM==LEXICAL ANALYZER
33020  REM  SEARCH QU$ FROM CHARACTER POSITION CP FOR ONE OF THE
33040  REM  CHARACTERS IN TM$.  ACCUMULATE THE NON SPACE
33060  REM  CHARACTERS BETWEEN THE STARTING POSITION AND THE
33080  REM  TERMINATING POSITION IN S$
33100  S$="":
       L=LEN(QU$):
       LS=LEN(TM$):
       CP=CP-1
33200  CP=CP+1:
       IF CP gt L THEN
          ER$="Syntax Error":
          GOTO 37000
33220  C$=MID$(QU$,CP,1):
       IF C$=" " GOTO 33200
33240  FOR TT=1 TO LS:
          IF C$=MID$(TM$,TT,1) THEN
             RETURN
33260  NEXT TT:
       S$=S$+C$:
       GOTO 33200

34000  REM==PARSE QUERY
34020  IF LEFT$(QU$,1)="." THEN
          OP(LQ)=10:
          NL=LQ-1:
          RETURN
34040  REM  INITIALIZE
       QU$=QU$+":":
       CP=1:
       ER$="":

34060  REM--LEVEL NO
       TM$="abcdefghijklmnopqrstuvwxyz":
       GOSUB 33000:
34080  IF S$="" THEN
          ER$="Level No missing":
          GOTO 37000
34100  LN=VAL(S$)

34120  REM--FIELD/LOGICAL OPERATOR
       TM$="/=⅛⅜:":
       GOSUB 33000:
34140  IF S$="" THEN
          ER$="Field missing":
          GOTO 37000
34160  IF C$ ne ":" GOTO 34300
34180  FOR OP=7 TO 9
34200     IF S$ ne OP$(OP) THEN
          NEXT OP:
          ER$="Illegal Logical Operator":
          GOTO 37000
34220  GOTO 34500
```

```
34300  REM--DOES FIELD EXIST?
34320  FOR F=1 TO NF:
          IF S$ ne FL$(F) THEN
          NEXT F:
          ER$="Field not found":
          GOTO 37000

34340  REM--RELATIONAL OPERATOR
       TM$="0123456789":
       GOSUB 33000:
34360  FOR OP=1 TO 6
34380     IF S$ ne OP$(OP) THEN
          NEXT OP:
          ER$="Illegal Relational Operator":
          GOTO 37000
34400  TV(LQ)=VAL(MID$(QU$,CP)):
       F(LQ)=F
34500  OP(LQ)=OP:
       LN(LQ)=LN:
       RETURN

36000  REM==DISPLAY QUERY LINE
36120  DL$=RIGHT$(STR$(100+LQ),2):
       IF OP(LQ)=0 GOTO 36300
36140  REM  END OF QUERY?
       IF OP(LQ)=10 THEN
          DL$=DL$+" .":
          GOTO 36300:
36160  REM  LEVEL
       DL$=DL$+LEFT$(B$,LN(LQ)*3)+RIGHT$(STR$(100+LN(LQ)),2)+" ":
36180  REM  FIELD
       IF OP(LQ) lt 7 THEN
          DL$=DL$+FL$(F(LQ)):
36200  REM  OPERATOR
       DL$=DL$+OP$(OP(LQ)):
36220  REM  VALUE REQUIRED
       IF OP(LQ) lt 7 THEN
          DL$=DL$+STR$(TV(LQ)):
36300  SYS TM[POKE ,0,LQ+1,1,DL$]
36320  RETURN

37000  REM==ERROR HANDLER
37020  SYS TM[POKE ,0,1,1,EM$]
37040  SYS TM[RESTORE]

40000  REM==GET A QUERY
40020  IF OP(1)=10 GOTO 40100
40040  SYS TM[POKE ,0,23,1,"New Query:"]
40060  SYS TM[INPUT ,23,13,0,QU$,1,1]:
       IF QU$ ne "y" AND QU$ ne "n" GOTO 40060
40080  IF QU$="n" GOTO 41000
```

```
40100  REM--INPUT THE NEW QUERY
40120  FOR LQ=1 TO MQ:
          SYS TM[SAVE ]
40130     S$=RIGHT$(STR$(LQ+100),2):
          SYS TM[POKE ,1,23,1,S$+":"]
40140     SYS TM[INPUT ,23,5,0,QU$,1,30]
40160     REM  PARSE
          GOSUB 34000:
          IF ER$ ne "" GOTO 40140:
40180     REM  DISPLAY
          GOSUB 36000:
40200     IF OP(LQ) ne 10 THEN
          NEXT LQ:
          SYS TM[POKE ,0,23,1,"Query too long"]:
          GOTO 40100
40220  NL=LQ-1

41000  REM==ALTER QUERY
41020  SYS TM[SAVE ]:
       SYS TM[POKE ,0,22,1,"Any Alterations:"]
41040  SYS TM[INPUT ,22,18,0,QU$,1,1]:
       IF QU$="n" THEN
          RETURN
41060  IF QU$ ne "y" GOTO 41040
41080  SYS TM[POKE ,0,23,1,"Line & Mods    :"]:
       SYS TM[INPUT ,23,18,0,QU$,1,32]
41100  LQ=VAL(LEFT$(QU$,2)):
       QU$=MID$(QU$,3):
       GOSUB 34000
41120  IF ER$="" THEN
          GOSUB 36000
41140  GOTO 41000

42000  REM==SET PARAMS FOR LOGICAL OPERATORS

42020  REM--LIST POINTER
       LS=0:
42040  FOR LQ=NL TO 1 STEP -1
42060     IF OP(LQ) lt 7 GOTO 42200
42080     REM  FROM POSITION
          F(LQ)=LS+1:
42100     REM  LOOK FOR ALL THE LINES WHICH ARE ONE
42120     REM  MORE THAN THE CURRENT ONE
42140     LR=LN(LQ)+1:
          FOR CL=LQ+1 TO NL:
             IF LN(CL)=LR THEN
                LS=LS+1:
                LI(LS)=CL
42160        REM  STOP WHEN THE LEVEL IS SAME OR LESS
             IF LN(CL) gt LN(LQ) THEN
             NEXT CL:
42180     REM  MARK THE 'TO' POSITION
          TV(LQ)=LS:
42200  NEXT LQ:
       RETURN
```

```
44000  REM==PROCESS QUERY
44020  REM  NUMBER SATISFIED
       SA=0:
44040  REM  CURRENT DISPLAY LINE
       DL=0:
44060  OPEN 2,8,2,"0:query.dt,s,r":
       OPEN 3,8,3,"@0:query.rs,s,w"

44100  REM--READ RECORD
       GOSUB 21000:
       IF ST ne 0 GOTO 44200:

44120  REM--IF OK THEN PRINT IT
       GOSUB 20000:
       IF R(1) THEN
          GOSUB 22000:
          SA=SA+1:
44140  GOTO 44100
44200  DL$="":
       GOSUB 23000

44220  REM--TERMINATING MESSAGE
       DL$=STR$(SA):
       IF SA=0 THEN
          DL$="No":
44240  DL$="    "+DL$+" record(s) satisfied":
       GOSUB 23000
44260  CLOSE 2:
       CLOSE 3
44280  RETURN

50000  REM==INITIALIZATION
50020  B$=" "
50030  C$="":
       CL=0:
       CP=0
50040  DF=0:
       DL=0:
       DL$="":
       DV=0
50050  ER$=""
50060  F=0:
       FW=12
50080  HE$=""
50120  L=0:
       LN=0:
       LQ=0:
       LR=0:
       LS=0
50130  MQ=20
50140  NF=6:
       NL=0
50150  OP=0
50160  PM=20
50170  QU$=""
```

```
50180  R=0:
       RS=0
50190  S$="":
       SA=0
50200  TM=6*4096:
       TT=0:
       TM$=""

51000  REM--ARRAYS
51060  DIM F(MQ),FL$(NF),FL(MQ)
51120  DIM LI(MQ),LN(MQ)
51150  DIM OP(MQ),OP$(10)
51180  DIM R(MQ),RC(NF)
51200  DIM TV(MQ)
52000  FOR I=1 TO 5:
          B$=B$+B$:
       NEXT I
52020  PRINT "[scr.clr]"
52040  FOR I=1 TO 10:
          READ OP$(I):
       NEXT I
52060  DATA "⅛","=","⅜","/⅛","/=","/⅜","and","ior","xor","."
52080  FOR I=1 TO NF:
          READ FL$(I):
          HE$=HE$+RIGHT$(B$+FL$(I),FW):
       NEXT I
52100  DATA "weight","age","height","eyes","hair","health"

52120  REM--SET TOP OF SCREEN IN CASE ERROR COMES ALONG
       POKE 224,1:
59999  RETURN
63998  REM          SCRATCH"query",D0:DSAVE"query",D0:VERIFY"*",8
63999  REM          SCRATCH"query",D1:DSAVE"query",D1:VERIFY"*",8
```

Appendix 6.2: Data

weight	age	height	eyes	hair	health
83	34	202	17	12	4
59	72	199	7	5	5
94	55	161	12	19	6
85	73	265	16	19	2
91	68	267	11	9	2
69	24	237	7	17	4
69	65	154	12	4	4
97	23	224	16	15	3
64	52	242	2	7	3
96	34	215	2	14	4
65	74	171	14	11	3
73	66	241	10	14	6
83	43	173	4	1	6
52	59	207	12	5	5
64	40	221	16	7	2
72	56	238	2	7	4
84	50	242	19	2	6
79	61	225	7	16	1
63	31	204	7	14	4
62	70	193	10	9	1
95	35	238	10	1	1
54	47	163	15	3	1
70	34	200	9	5	5
74	67	192	2	2	2
94	43	252	1	11	4
50	68	241	6	17	6
76	47	231	2	19	6
71	61	209	4	10	5
78	35	182	14	15	4
73	25	175	9	3	2
70	32	183	3	7	4
80	55	151	19	7	4
87	23	198	14	12	3
88	38	226	3	6	5
73	39	234	19	18	2
53	30	238	6	3	5
54	36	236	13	19	4
63	53	167	3	10	6
70	25	262	18	12	4
79	64	161	16	17	5
91	74	259	3	6	6
84	23	174	19	14	4
90	56	227	13	7	1
80	58	183	11	4	3
60	68	155	10	14	4
76	67	233	14	1	3
74	30	218	17	14	3
60	53	190	9	7	6
61	67	217	14	14	2
56	66	207	13	15	2

Appendix 6.3: Results

The following query was performed on the data in Appendix 6.2:

```
01 and
   02 and
      03 height gt 200
      03 height lt 250
   02 age gt 60
```

The results of the query follow:

weight	age	height	eyes	hair	health
73	66	241	10	14	6
79	61	225	7	16	1
50	68	241	6	17	6
71	61	209	4	10	5
76	67	233	14	1	3
61	67	217	14	14	2

6 record(s) satisfied

End of Appendix

REFERENCES

If you do not understand a particular word in a piece of technical writing, ignore it: the piece will make perfect sense without it

- Mr Cooper's Law

ACM = Association of Computing

CPUCN = Commodore PET Users' Club Newsletter

Alagic S., Kulenovic A., 1981

Relational Database Pascal Interface

Computer Journal 24, 112-117

Alcock D., 1978

Illustrating Basic (A Simple Programming Language)

Cambridge University Press

American National Standards Institute, 1978

Programming Language FORTRAN

ANSI X3.9, New York

Anliker M., 1980

Computer Assisted Image Analysis in Medicine

In: Data Base Techniques for Pictorial Applications

Springer-Verlag, 365-368

Arisawa M., Iuchi M., 1980

Debugging Methods in Recursive Standard Fortran

Software Practice & Experience 10, 29-44

Avison D.E., 1981

Techniques of Data Analysis

Computer Bulletin [September], 9-11

Bacon M.D., Bull G.M., 1973

Data Transmission

MacDonald & Co (Publishers) Ltd.

Barron D.W., 1968

Recursive Techniques in Programming

MacDonald & Co (Publishers) Ltd.

Barron D.W., 1977

An Introduction to the Study of Programming Languages

Cambridge University Press

Bayer R., McCreight C., 1972

Organization and Maintenance of Large Ordered Indexes

Acta Informatica 1, 173-189

Belady L.A., 1966

A study of replacement algorithms for a Virtual Store Computer

IBM Systems Journal 5, 78-101

Bell J.R., 1970

The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering

Comm. ACM 13, 107-109

Best P.J., 1980

Personal Communication

Bezier P., 1972

Numerical Control - Mathematics and Applications

J. Wiley & Sons, London

Bird R.S., 1977

Notes on Recursion Elimination

Comm. ACM 20, 434-439

Bloch A., 1977

Murphy's Law and other reasons why things go wrong

Price/Stern/Sloan Publishers Inc., Los Angeles

Boothroyd J., 1967

Chebyschev Curve Fit

Comm. ACM 10, 801-803

Boyer R.S., Moore J.S., 1977

A Fast String Searching Algorithm

Comm. ACM 20, 762-772

Brown P.J., 1972

Re-creation of Source Code from Reverse Polish Form

Software Practice and Experience 2, 275-278

Butterfield J., Middleton D., 1980

Basic 2/Basic 4 ROM Comparison

CPUCN 3:1, 41-46

Butterfield J., Russell J., 1979

New/Old ROM Memory Map

CPUCN 2:3, 10-16

Cadwell J.H., Williams D.E., 1961

Some Orthogonal Methods of Curve and Surface Fitting

Computer Journal 3, 420

Chasen S.H., 1978

Geometric Principles and Procedures for Computer Graphics Applications

Prentice-Hall, New Jersey

Chatfield C., 1970

Statistics for Technology

Chapman & Hall

Chen P., 1977

The Entity-Relationship Approach to Database Design

Database Management

Clenshaw C.W., 1960

Curve Fitting with a Digital Computer

Computer Journal 2, 170

Colin A.J.T., 1963

Note on coding Reverse Polish Expressions for single address computers with one accumulator

Computer Journal 6, 57-68

Comer D., 1979

The Ubiquitous B-Tree

Computing Surveys 11, 121-137

Commodore Business Machines, 1980

Commodore Assembler Development Package Users' Manual

Commodore Business Machines, 1980

CBM Dual Drive Floppys, Model 2040, 3040, 4040, 8050

P/N 320899

Commodore Business Machines, 1979

CBM 3032 Professional Computer Users' Manual

P/N 320856-3

Connecticut Microcomputer, 1980

PETSET1 Instruction Manual

Cooke J.A., 1979

IEEE Bus Handshake Routine

CPUCN 1:4

Cooper B.E., 1968

Basic Subroutine for the input of numbers, words and special characters

Computer Journal 11, 157-168

Cooper J., 1977

The minicomputer in the Laboratory with Examples using the PDP-11

J. Wiley & Sons, London

Coyle F.T., 1971

The hidden speed of ISAM

Datamation [June], 48-49

Cronin D.E., Brandon J.P., 1972

A High Speed Computer-to-Computer Data Link

Software Practice and Experience 2, 173-186

Data General Corporation, 1977

Text Editor User's Manual

093-000018-09

Data General Corporation, 1978a

Fortran IV User's Manual

093-000053-09

Data General Corporation, 1978b

RDOS/DOS Command Line Interpreter User's Manual

093-000109-01

Day A.C., 1970

Full Table Quadratic Searching for Scatter Storage

Comm. ACM 13, 481

De Morgan R.M., Hill I.D., Wichmann B.A., 1976

Modified Report on the Algorithmic Language Algol 60

Computer Journal 19, 364-379

Denning P.J., 1970

Virtual Memory

Computing Surveys 2, 153-189

Digital Equipment Corporation, 1976a

MUMPS-11 Language Reference Manual

DEC-11-MMLMA-D-D

Digital Equipment Corporation, 1976b

MUMPS-11 Programmer's Guide

DEC-11-MMPGA-D-D

Dowson P.L., 1980

Wordcraft 80 Reference Guide

Commodore Business Machines

Earnshaw R.A., 1980

Line Tracking for Incremental Plotters

Computer Journal 23, 46-52

Elmore R.W., Agarwal K.K., 1980

An Information Retrieval System

Byte 5:10, 114-150

Evans J.M., 1979

Solving the Interface Transfer Problem

Practical Computing 2:11, 87-89

Fisher E., Jensen C.W., 1980

PET and the IEEE-488 Bus (GPIB)

McGraw-Hill Inc.

Fisher R.A., Yates F., 1963

Statistical Tables for Biological, Agricultural & Medical Research

Oliver & Boyd, Sixth Edition

Forsythe G.E., 1957

Generation and use of Orthogonal Polynomials for Data Fitting with a Digital Computer

Journal SIAM 5:2, 74-88

Galimberti R., Montanari U., 1969

The Perspective Representation of Functions of Two Variables

Comm. ACM 12, 206-211

Gazioglu K., Condemi J., Kaltreider N.L., Yu P.N., 1968

Study of Forced Vital Capacity and Maximal Expiratory Flow Volume Curves in Obstructive Lung Diseases

American Review of Respiratory Diseases 98, 857-867

Golub G.H., Wilkinson J.H., 1966

Note on the iterative refinement of Least Squares Solution

Numerische Mathematik 9, 139

Gould I.H., 1971

Communications between Unrelated Computers

Institute of Computer Science, University of London

Hancox A.J., 1980

Personal Communication

Hebditch D.L., 1975

Data Communications: An Introductory Guide

Paul Elek (Scientific Books) Ltd.

Henry D., [undated]

Analog to Digital Conversion - its Principles, Practices and Pitfalls

Dept. of Anaesthetics, University of Birmingham

Henry D., [undated]

The Use of a PDP 9 Computer as a Data Aquisition Device, to be used with an FM Magnetic Tape Recorder

Dept. of Anaesthetics, University of Birmingham

Hoeschele D.F., 1968

Analogue-Digital/Digital-Analogue Conversion Techniques

Wiley, New York

Hopgood F.R.A., 1968

A solution to the table overflow problem for hash tables

Computer Bulletin 11, 297

Horspool R.N., 1980

Practical Fast Searching in Strings

Software Practice and Experience 10, 501-506

Hyatt R.E., Black L.F., 1973

The Flow Volume Curve: A Current Perspective

American Review of Respiratory Diseases 107, 191-199

Ingram R.H., Schilder D.P., 1966

Effect of Gas Compression on Pulmonary Pressure, Flow and Volume relationship

Journal of Applied Physiology 21, 1821-1826

James G., 1976

Revised Methods of Lung Function Testing

Clinical Investigation Unit, Dudley Road Hospital

Johnson L.R., 1961

Indirect Chaining Method of Addressing on Secondary Keys

Comm. ACM 4, 218-222

Kilburn T., Edwards D.B.G., Lanigan M.J., Summer F.H., 1962

One Level Storage System

IRE Trans. on Electronic Comp. 11:2, 233-235

Kirkman J., 1971

What is Good Style for Engineering Writing

Institute of Chemical Engineers, London

Knuth D.E., 1969a

Semi-Numerical Algorithms

In: The Art of Computer Programming

Addison-Wesley

Knuth D.E., 1969b

Sorting and Searching

In: The Art of Computer Programming

Addison-Wesley

Kubert B., Szabo J., Giulieri S., 1968

The Perspective Representation of Functions of Two Variables

Journal ACM 15, 193-204

Landan L.I., Hill D.J., Phelan P.D., 1973

Factors determining the shape of Maximum Respiratory Flow Volume Curves in Childhood Asthma

Australian & New Zealand Journal of Medicine 3, 557-564

Levine S.T., 1973

Instrument Interfacing can be done - but it takes a bit of doing

Electronic Design 24 [Nov 22]

Lloyd P., 1966

Graph Plotter

Comm. ACM 9, 88

Logica Ltd., 1979a

Rapport: Designing and Using a Database

Q9790Q

Logica Ltd., 1979b

Rapport: Interactive Query Language User's Manual

T5790Y

Logica Ltd., 1979c

Rapport: User Manual

U4B019

MacKinney J.G., 1960a

Least Squares Fit by Orthogonal Polynomials

Comm. ACM 3, 604

MacKinney J.G., 1960b

Polynomial Transformer

Comm. ACM 3, 604

MacMillan D.B., 1961

Remarks on Least Squares Fit by Orthogonal Polynomials

Comm. ACM 4, 544

Makinson G.J., 1967

Remarks on Least Squares Fit by Orthogonal Polynomials

Comm. ACM 10, 293

Martin J., 1973

Design of Man Computer Dialogues

Prentice Hall

McConalogue D.J., 1970

A Quasi-intrinsic Scheme for passing a Smooth Curve through a Discrete Set of Points

Computer Journal 13, 392-396

McIlroy M.D., 1963

A Variant Method of File Searching

Comm. ACM 6, 101

Meerwall E. von, 1976

A flexible, All Purpose Curve Fitting Program

Computer Physics Communications 11, 211-219

Morris R., 1968

Scatter Storage Techniques

Comm. ACM 11, 38-43

MOS Technology Inc., 1976a

MCS 6500 Microcomputer Family Hardware Manual

MOS Technology Inc., 1976b

MCS 6500 Microcomputer Family Programming Manual

Muir D., 1980

PET and the IEEE Bus

CPUCN 3:5, 22-28

Newman N., Lang T., 1976

Documentation for Computer Users

Software Practice and Experience 6, 321-326

Nievergelt J., 1973

Binary Search Trees and File Organization

Computing Surveys 6, 195-207

Open University, 1972

Chebyschev Approximation

Open University Press, M201:27

Pagan F.G., 1976

On Interpreter oriented definitions of programming languages

Computer Journal 19, 151-155

Pagan F.G., 1979

Algol 68 as a metalanguage for denotational semantics

Computer Journal 22, 63-66

Pagan F.G., 1981

A Style for writing the syntactic portions of complete defintions of programming languages

Computer Journal 24, 143-145

Parkhurst R.J., 1968

Program Overlay Techniques

Comm. ACM 11, 119-125

Payne J.A., 1970

An Automatic Curve Fitting Package

In: Numerical Approximation to Functions and Data

University of London, 98-106

Pearson E.S., Hartley H.O., 1966

Biometrika Tables for Statistics

Third Edition, Cambridge University Press

Powell M.J.D., 1967

On the Maximum Errors of Polynomial Approximations defined by Interpolation and by Least Squares

Computer Journal 9, 404

Price C.E., 1971

Table Lookup Techniques

Computing Surveys 3, 49-65

Prowse P., 1980

The Database Approach

Computer Journal 23, 9-12

Quitzow K.H., Klopprogge M.R., 1980

Space Utilization and Access Path Length in B-Tree

Information Systems 5, 7-16

Reingold E.M., 1981

A comment on evaluation of Polish postfix expressions

Computer Journal 24, 288

Ricci D.W., Nelson G.E., 1974

Standard Instrument Interface Simplifies System Design

Electronics 47:23 [Nov 14], 95-106

Riche P.J. le, 1969

Curve Plotting Procedure

Computer Journal 12, 291

Rogers B., 1980

Fast Fourier Transforms

Practical Computing [Dec], 91-93

Rogers D.F., Adams J.A., 1976

Mathematical Elements for Computer Graphics

McGraw-Hill, New York

Rohl J.S., 1977

Converting a class of recursive procedures into non-recursive ones

Software Practice and Experience 7, 231-238

Ruckdeschel F., Krinsky J.A., 1981

A Simple Approach to Data Smoothing

Byte 6:3, 262-298

Santoni A., 1977

What's wrong with 488? Not Much, but ...

Electronic Design 24 [Nov 22], 48-51

Schay G., Spruth W.G., 1962

Analysis of a File Addressing Method

Comm. ACM 5, 459-462

Sernadas, A., 1981

SYSTEMATICS: Its syntax and Semantics as a Query Language

Computer Journal 24, 125-129

Sharp J.A., 1980

Data Oriented Program Design

ACM Sigplan Notices, 15:9, 44-57

Small Systems Engineering [undated]

The IEEE-488 to RS-232 Bidirectional Serial Interface Type B

Operating Instructions

Smith J.M., Smith D.C.P., 1977

Database Abstractions: Aggregation

Comm. ACM 20, 405-413

Spacek T.R., 1972

A proposal to establish a vitrual memory via writeable overlays

Comm. ACM 15, 421-462

Steele G.L., 1977

Arithmetic Shifting Considered Harmful

ACM Sigplan Notices 12:11, 61-69

Strong J., et. al., 1958

The problem of programming communications with changing machines: A proposed solution

Comm. ACM 1:8, 12-18, Comm. ACM 1:9, 9-15

Strutt A.C.R., Hobbs K.W., 1980

Data Input to Commodore PET via a Parallel to Serial Converter

CPUCN 3:3, 21-23

Tassel D. van, 1978

Program Style, Design, Efficiency, Debugging and Testing

Second Editon, Prentice-Hall

Taylor Wilson Systems Limited, 1980

Commlink

TNW Corporation, 1979

TNW 3000 Serial Interface User's Manual

Wegner P., 1968

Programming Languages, Information Structures and Machine Organization

McGraw-Hill

White J.W., Ripley G.D., 1977

How Portable are Minicomputer Fortran Programs

Datamation 23 [July], 105-107

Whiteney G., 1969

An Extended BNF for specifying synatx declarations

Spring Joint Computer Conference, 801-802

Wild R., 1971

Production and Operations Management

Holt Rinehart and Winston Ltd.

Willis C., Liddiard L., 1972

A note on packing and unpacking of bytes

Software Practice and Experience 2, 401-402

Wirth N., 1976

Algorithms + Data Structures = Programs

Prentice-Hall

ACKNOWLEDGEMENTS

Thinly Sliced Cabbage - Cole's Law

Last but not least, this thesis would be incomplete without an acknowledgement of thanks to the following [in alphabetical order]:

- Department of Production Technology and Production Management and the Computer Centre of Aston University for providing the computing and graph plotting facilities.
- Dr Anthony Hancox of the Clinical Investigation Unit at Dudley Road Hospital in Birmingham for pen recorder outputs of the Pressure Concentration Curve and up-to-date information about the happenings at the CIU.
- Dr Leslie Hazelwood for his marvellous lectures on curve fitting.
- The late Ronald Keeble for suggestions on how a mathematician would attempt to fit the pressure concentration curve.
- Eric Langton for hardware assistance and amusing anecdotes.
- Trevor Law for the negotiating the loan of his company's Daisy-Wheel printer and for allowing the use of the Departmental photocopying facilities.
- Dr Robert Loughnane, without whom I never would have ventured into Biomedical Engineering.
- Dr Douglas Love for Commlink and his interpretation of the fuzzy sections of the thesis regulations.
- Rajendran Narayanan for memorable arguments on information retrieval and for his help in bypassing the security used by the word processing system, Dr Fat Crow.
- S. Sabaratnam for the loan of his computer for a few weekends.
- My supervisor, Dr David Scrimshire for initial guidance, proof reading, obtaining some of the vital equipment for the project, and negotiating the use of the Departmental Daisy-Wheel printer.
- N. Williams for the loan of the signal generator, oscillator and the oscilloscope.

Cheah Ui Poh