THE USE OF ALPHANUMERIC DISPLAY TERMINALS

A thesis presented in partial fulfilment of the requirements for the degree of Master of Arts in Computer Science at Massey University

Patrick John Barrer

1975

Acknowledgements

I should like to thank my supervisors, Bob Doran and Professor Tate, for their help and encouragement. Bob Doran originated the Genisys project, designed the General language and implemented Genisys's data structures.

I should also like to thank Chris Freyberg for his valuable help in interfacing the VT05 to the B6700 and Ted Drawneck for reading my thesis and for his helpful comments.

Abstract

A functional description of alphanumeric display terminals is given and then the four main classes of interactive systems that use alphanumeric display terminals; enquiry/response systems, interactive programming systems, data entry and computer-aided instruction are briefly described.

Some programming considerations where using alphanumeric display terminals with interactive systems are mentioned and then a "virtual" alphanumeric display terminal is introduced.

Finally as an example, the program entry phase of the Genisys system is described.

Preliminary Note

Throughout this thesis the term alphanumeric display terminal will from time to time be abbreviated to ADT.

TABLE OF CONTENTS

		Page
	Acknowledgements	i
	Abstract	ii
	Preliminary Note	iii
CHAPTER 1	A Functional Description of Alphanumeric Display Terminals	1
contents:	The Processor-Terminal Connection	1
	The Components of a typical ADT	3
	The Transmit and Receive Interface	3
	The Display Screen	4
	The Memory	6
	The Character Generating Subsystem	6
	The Keyboard	7
	The Control Logic	8
	Operating Modes	9
	Full-Duplex and Half-Duplex Modes	9
	Block and Conversational Modes	10
	Receive and Transmit Modes	11
	Additional Control Functions	12
	Scrolling and Paging	12
	Editing Capability	13
	Protected Fields	14
CHAPTER 2	Interactive Systems which use Alphanumeric Display Terminals	16
contents:	Enquiry/Response Systems	18
	Airline Reservation Systems	19
	The Medical Information System at El Camino Hospital, California	19
	Interactive Programming Systems	21
	CANDE, an example of a general purpose editor	22
	Single Language Systems	24
	The Interactive Programming Support System	25
	Emily	26

Page

The Automated Computer Science Education System 27

Data Entry 28

Conventional Data Entry Using ADTs 28

Form Filling 29

Computer Aided Instruction 30

The COPI system 31

The Socratic approach 32

CHAPTER 3 Some Programming Considerations when using ADTs with Interactive Systems 33

contents:

Dialogue Design 33

Passive Systems 34

Active Systems 35

Screen Management 37

Use limitations 37

Size limitations 38

Operating Modes 41

Conversational full-duplex mode 41

Half-duplex block mode 42

Conversational half-duplex mode 42

Hardware Requirements 42

CHAPTER 4 Virtual Alphanumeric display terminals 46

contents:

Some reasons for implementing virtual terminals 46

An example - a software interface for the DEC VT05 ADT 48

CHAPTER 5 An Example - Constructing Programs with Genisys 51

contents:

Screen Management 56

Dialogue Design 57

Line Entry 59

Using Genisys 59

Summing up Genisys 64

Page

APPENDIX I Summary of ADT characteristics 65

APPENDIX II Survey of ADTs available in New Zealand 67

APPENDIX III The Digital Equipment Corporations VT05 ADT 80

APPENDIX IV General's syntax 82

APPENDIX V Description of Buildprogram 83

Bibliography and References 93

List of Figures

		Between Pages
Figure 1.1	The ASCII 67 character code	2-3
Figure 1.2	The main components of a typical ADT	3-4
Figure 1.3	Some examples of dot matrix characters	5-6
Figure 3.1	A hypothetical menu	36-37
Figure 3.2		36-37
Figure 5.1		56-57
Figure 5.2		56-57
Figure 5.3		60-61
Figure 5.4		60-61
Figure 5.5		60-61
Figure 5.6		60-61
Figure 5.7		60-61
Figure 5.8		60-61
Figure 5.9		60-61
Figure 5.10		60-61
Figure 5.11		61-62
Figure 5.12		61-62
Figure 5.13		62-63
Figure 5.14		62-63
Figure 5.15		62-63
Figure 5.16		62-63
Figure 5.17		63-64
Figure 5.18		63-64
Figure 5.19		63-64
Figure 5.20		63-64

CHAPTER 1

A Functional Description of Alphanumeric Display Terminals

An alphanumeric display terminal is one of many devices that allow a person to communicate with a computing system. Typically such devices have a keyboard which the operator may use to enter messages and a screen where alphanumeric characters may be displayed.

More and more ADTs are being used with computing systems, and a large variety of different models is now available; indeed, Auerbach's "Guide to Alphanumeric Display Terminals" lists more than two hundred and fifty different terminals [Datamation May 1975, p30.]

This chapter described some of the basic features of these terminals. The discussion takes place under the following headings: the Processor-Terminal Connection; the Components of a typical ADT; Operating Modes; Additional Control Functions.

The Processor-Terminal Connection

Although most alphanumeric display terminals can be operated off-line, their basic use is to transfer information between the human operator and the computer. To accomplish this a terminal has to be physically linked to the computing system. This section is concerned with this link.

A terminal and processor communicate using characters. A character is a unique piece of information that can be represented in different ways. On the screen of the terminal the character is represented by a single graphic symbol but when held in the memory of the terminal and of the computer a character is represented as a set of bits. Because the physical lines which connect the terminal to the processor are, like the elements of memory, best limited to two states, characters transferred between processor and terminal are encoded as sets of bits. Since terminals are

$b_7 b_6 b_5$ \ $b_4 b_3 b_2 b_1$	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	SOH	DC 1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	×	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	−	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	_	o	DEL

Fig1.1 ASCII 67 character code

ASCII Control characters

NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC 1	Device Control 1
STX	Start of Text	DC 2	Device Control 2
ETX	End of Text	DC 3	Device Control 3
EOT	End of Transmission	DC 4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell (audible signal)	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tab	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertifcal Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
		DEL	Rubout

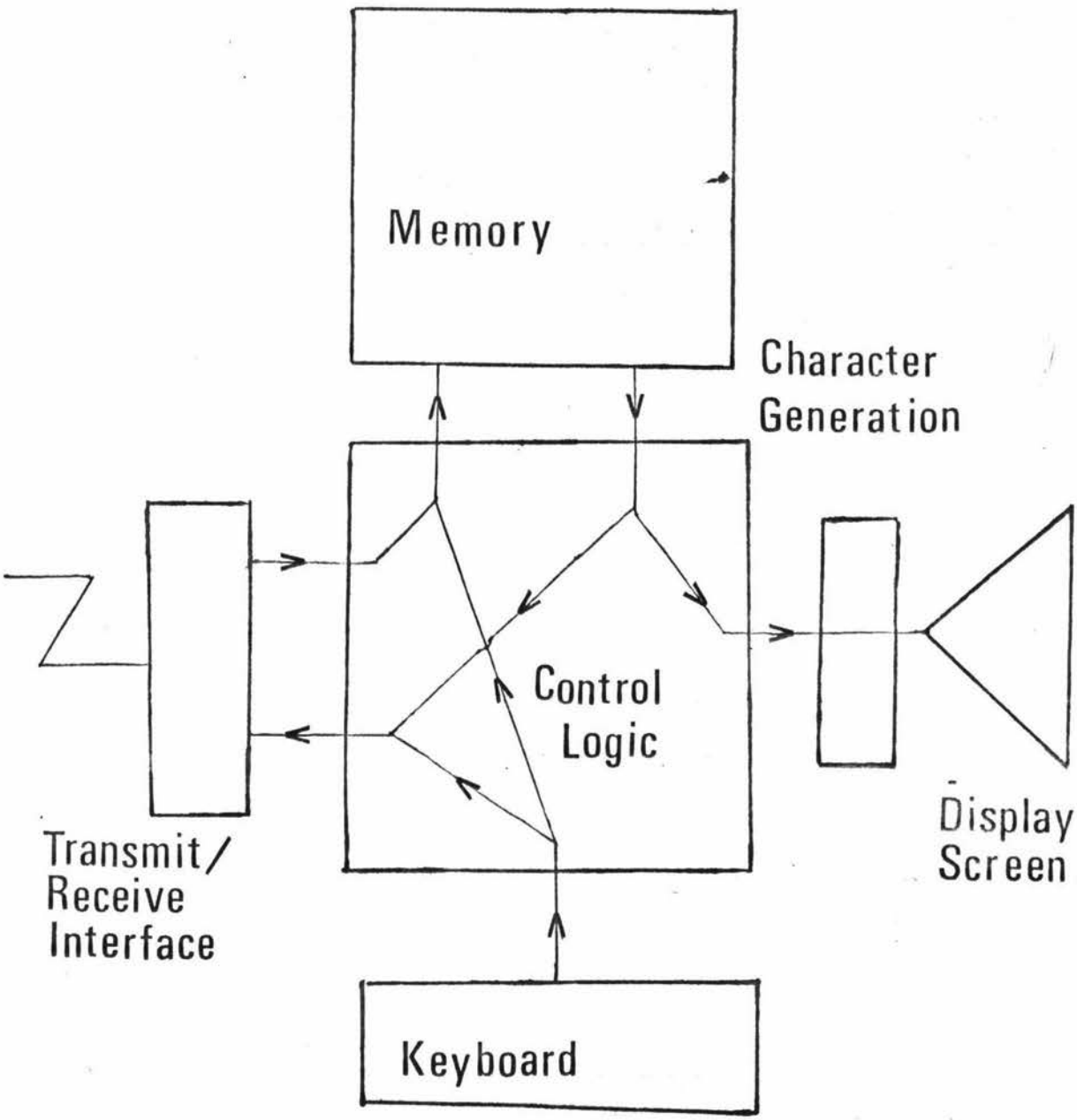
figure 1.1 (continued)

designed for use with more than one computer, a standard code, the American Standard Code for the Interchange of Information 1967 (ASCII 67) is frequently used to represent the characters. This is a seven bit code which therefore allows one hundred and twenty-eight different characters.

The transfer of information between terminal and processor is always in the form of a message sent one character at a time. The link on which the message is sent may take two forms. The first provides direct data paths for each bit in the character representation, each bit thereby being transmitted simultaneously. With such a link, data can be transferred at speeds that can approach two million characters per second. More commonly, however, only one data path is provided by the link. The characters must be transmitted serially; that is, bit by bit. This is done at switch-selectable rates generally ranging from one hundred and ten bits per second (or about ten characters per second) to a maximum of nine thousand six hundred bits per second (or about a thousand characters per second). Some newer terminals now offer twice this rate. The lower rates are provided only to give plug-in compatability with teletypes, that is, to use the lower grade teletype lines.

Serial characters are transmitted either synchronously or asynchronously. Synchronous transmission means that the characters comprising the message are transmitted at regular intervals. The message is preceded by a "header", a special set of characters which describe the following message, the last character of the message is a special end of text character. Asynchronous transmission means that the characters can be sent at either irregular intervals or at regular intervals, each character preceded by a special start bit (or bits) to signal the start of that character, and followed by a stop bit to signal the end of the character.

Data communication lines can be either half-duplex or full-duplex. Half-duplex lines allow transmission only in one direction at one time. Full-duplex lines allow



Main Components of an ADT

simultaneous transmission in both directions.

Half-duplex lines have problems of traffic control since traffic can be allowed in only one direction. Further problems of traffic control arise when terminals, as often happens, are forced to share a line. These problems are resolved by the application of line disciplines, predetermined methods of "rationing" line use to allow the orderly transference of messages. Two brief examples may be given. "Polling" is a line discipline in which the processor polls each terminal in some predetermined pattern to see if there is a message ready for transmission from it. If no terminal has such a message then the processor can transmit to any or all terminals any messages it has. "Contention" is a line discipline in which the terminal and processor contend for the line, each transmitting its messages at regular but different intervals and continuing until acknowledgement has been received from the other end of the line that the message has been received.

The Components of a typical ADT

An alphanumeric display terminal may be viewed as consisting of six basic components. These are the display screen, the memory, the character generating system, the transmit and receive interface and the control logic. Each of these components provides a subheading for this section in which they are further described.

The Transmit and Receive Interface

This interface superintends the transfer of information between the computer and the terminal. The complexity of the interface depends on the line disciplines under which it is designed to operate. Many are very simple - they transmit from the terminal the appropriate character when any key is depressed or else they transmit a block of characters out of the memory when the transmit key is depressed. Characters received from the computer are transferred to the control logic which enters them into the memory if they are displayable or else responds to them if

they are control characters. Such a terminal is intended to have its own line. Where a terminal must share a line, the terminal must be addressable, that is, the interface must be capable of discriminating among messages to determine those which are intended for it. Messages may have to be acknowledged and complicated dialogues may have to take place before a terminal can transmit a message. For example, where a polling line discipline is operating and a terminal has a message to send, it must first wait till it is polled by the computer, then indicate that a message is ready for transmission, and then send the message. Under a contention line discipline, the terminal may have to retransmit the message at regular intervals until an acknowledgement is received. In general the interfaces usually provided for ADTs do not allow particularly sophisticated line disciplines, although there are exceptions (often provided as extras at an additional cost).

The Display Screen

Two types of display screen have been used with ADTs. In most cases the display screen is a cathode ray tube. In a cathode ray tube an electron gun directs a beam of electrons towards a phosphor-coated screen which glows when struck by the beam.

Three different methods are used to draw the characters on the screen. The electron beam may be forced to pass through a mask in the shape of the desired character which is then projected onto the screen. Another method is to construct the characters from small strokes of the electron beam. Both these methods have largely been superseded by the dot matrix method which although it gives a slightly less satisfactory representation is much simpler to implement. The electron beam moves in a regular fashion across the screen moving from side to side and top to bottom. Although for most of its journey it is turned off, as it passes certain points it may be turned on, illuminating at that point a dot on the screen. Each displayed character is built from a matrix of these dots. For example in a 5x7

matrix (which is the most common arrangement) there are 35 dots from which characters are built (as in fig 1.3).

The number of displayable characters depends to some extent on the size of the matrix. A 5x7 matrix is about the smallest sufficient to display uppercase letters, digits and special characters. Lower-case letters can also be displayed with this matrix if the tails of p,q,g and j can be displayed in the two dot line gap between lines of characters. Matrices with more dots give more pleasing representations of the characters.

Some terminals have the ability to display characters with reversed contrast (that is all dots not normally illuminated are illuminated while the others are blanked), to blink between the normal and reversed representations, to display characters in different character fonts, or to display characters at different intensities or even colours.

At some particular point on the screen the current cursor position will be displayed. The cursor indicates the point at which a newly-entered character would be displayed on the screen. The cursor position is moved by the control logic, which keeps its location in special registers, usually storing the x and y displacements. The actual form the cursor takes when displayed depends on the terminal. It may be a blinking line underscoring the character in that position, a character being alternated with its reverse, or any other distinctive feature. Usually some form of blinking is involved so as to attract the user's attention.

Mention must be also made of a less usual form of screen technology just making its appearance. This is the gas plasma screen which consists of a large matrix of gas cells each of which can be selectively illuminated. Characters are built in the dot matrix fashion described previously. Each gas cell is filled with neon gas which can be made to glow. This effect lasts longer than the momentary glow of a phosphor when an electron strikes it. Because the duration of the illumination can be unpredictable, Burroughs "Selfscan" technology in its pioneering use of gas plasma screens refreshes the display constantly in a similar manner

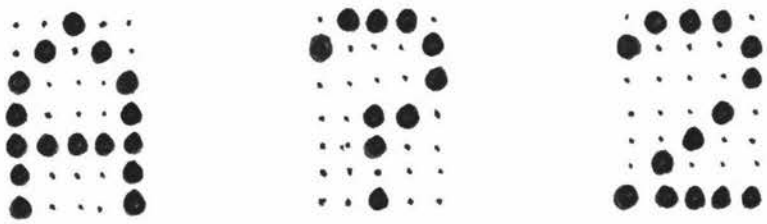


Fig 1.3

Some examples of dot matrix characters

to that used with a cathode ray tube.

Memory

Since the display must be constantly refreshed, each character displayed on the screen must be held in an internal form in permanent storage. This is the main memory of the terminal, sometimes referred to as the buffer. The memory is usually constructed from semi-conductor shift registers since the whole memory is being read serially all the time the device is working. The correspondence between screen and memory is determined in one of two basic ways. Most terminals let each position on the screen correspond to an unique position in the memory. A displayable character (which is often the space code) is always stored there and the corresponding character is displayed on the screen.

The second less common method derives from times when memory was relatively more expensive and sometimes shared between several terminals. A non-spatial memory stores all the character codes, excluding unnecessary space codes but including control character codes, in consecutive locations, the control characters being used to generate the actual position of the displayable characters. Since large portions of the screen are frequently blank, this uses less memory. It can also make some editing features easier.

The same memory may be used to drive several screens or it may be large enough to hold reserve pages for the same screen, each of which can be loaded from the computer and may be used to replace the current display very quickly.

Character Generation

Each character code is fetched at regular intervals of about a fiftieth of second from the main memory and used to refresh the display. The internal code must be changed into the character matrix. The easiest way to do this is to hold the matrix in a read only memory at an address produced from the numerical valve of the code. If the code is ASCII, this is facilitated by the fact that all the displayable characters are in numerical order starting from 20 Hex and differing by 1 - unlike the EBCDIC code for

example where there are gaps and control characters mixed among the displayable characters. The matrix is then used to control the illuminations of the particular dots which make up each character.

Thus, to display a character it is sufficient to enter the character code into the memory and as a result it will be displayed within a fiftieth of a second. Likewise alteration of characters or the cursor position on the screen merely involves the alteration of the memory or the special registers which hold the cursor position.

Keyboard

The keyboard usually resembles that of a teletypewriter, although some manufacturers provide options, usually in the number of control keys available or by the provision of separate numeric pads. The basic action of the keyboard is that when a key is depressed, a character is generated in a special register. This character is usually in the form in which it would be stored in the terminal's memory or transmitted to the processor. Shift and control keys are provided to extend the number of characters generated by the keyboard. In ASCII terminals, keyboards can commonly generate all 128 ASCII characters, the shift and control keys each changing one bit in the final character.

What happens to the character in the register depends on which of the various alternative modes the terminal is operating in. These modes are reviewed in more detail in a later section but their effect will be described briefly. If the terminal is in full-duplex mode, the character is transmitted to the computer. If the terminal is in half-duplex mode, then as well as being transmitted the character is also inserted in memory or if it is a control character, the control logic reacts appropriately. If the terminal is in block mode, the character is not transmitted, but only consumed locally. The reasons for these differing actions are explained later.

As well as keys which generate control and displayable characters, some ADTs also have function keys. Function

keys which are more commonly associated with graphics terminals generate characters which are neither displayable nor cause any action to be taken by the control logic but which, when transmitted to the computer, invoke some programmed response. A common example is the rubout or DEL key specifically provided for in the ASCII character set which can indicate to the program that the previous character is to be ignored. (This is a remnant from teletypes - with an ADT correction is better made by overwriting).

The Control Logic

The control logic performs certain actions to alter memory contents (and hence the display) and cursor position. It does this either automatically on receipt of ordinary, displayable characters from either the keyboard or the processor, or in response to special control characters. This control logic is usually hardwired or at least firm-wired (driven by a micro-program held in read only memory); however, there are some "intelligent" terminals which are programmable.

The basic automatic control function is the insertion of the received character into memory at the point indicated by the cursor and the advance of the cursor one space.

Another commonly provided automatic function is "wraparound" at the screen boundaries. If the cursor is at the end of one line and a new character is to be displayed then the cursor is advanced automatically to the beginning of the next line and the character displayed there. Wraparound may occur at all boundaries or only at the end of lines.

The basic control characters are used to move the cursor, either forward or backward one position, up or down one line, or to the start of the current line (carriage return), or back to the first position on the screen (usually called 'home"). Other necessary control characters cause the erasure of all or just part of the screen.

Less necessary, but still very useful, control functions allow the cursor to move larger distances. Tabbing, where the cursor is moved to the next predetermined position in a line on receipt of the tab control character, is provided on most terminals. The tab positions may be fixed (at multiples of eight character positions for example) or else may be preset by the programmer. Another very useful control feature available on a number of terminals is cursor addressing, where the cursor may be quickly transferred to any other position on the screen. This is generally done by the transmission of three characters, the first a special control character and the other two ordinary characters which are not displayed but whose numerical values give the line and position on that line to which the cursor is to be moved. Less frequently provided but sometimes useful is the ability for the processor to read out the cursor's current address. Some more sophisticated control features are also discussed in a later section in this chapter.

Operating Modes

Since no designers can be sure of the circumstances in which their terminal will operate, they frequently provide switch-selectable alternatives. The unknown factor which they must cater for is the nature of the connection between the terminal and the processor. Three different problems are solved by the provision as options of full-duplex and half-duplex modes, block and conversational modes and receive and transmit modes; and these constitute the subheadings under which each of these alternatives are discussed.

Full-Duplex and Half-Duplex Modes

Full-duplex and half-duplex modes are provided to give a terminal the capability to use both full-duplex and half-duplex lines. The mode is generally selected when the terminal is installed and it is unusual for the mode to be changed after this.

In full-duplex mode the keyboard is detached (logically not physically) from the rest of the terminal and must communicate with the display via the connected processor. This is, when a key is depressed, no character is displayed until it has been "echoed" by the processor. In effect all keys become function keys dependent on a programmed response for any result from the key being depressed. Normally the programmed response will be to "echo" or send back the character received so the user may see what is being typed. Such a system requires the terminal to be connected by a full-duplex line since characters must be sent continually in both directions.

In half-duplex mode the character is immediately entered into the terminal's memory (and so appears on the display) when the key is depressed so a half-duplex line may be used to connect the terminal. A full-duplex line may also be used to connect a terminal in half-duplex mode if it is felt that making the processor echo the characters is an unacceptable overhead or otherwise inconvenient.

Block and Conversational Modes

Unlike half and full duplex modes, block and conversational modes are often not provided as switch selectable alternatives on the same terminal since there is a philosophical disparity between them. Hence the terminal must be selected according to the mode in which it is designed to operate.

Conversational mode terminals (sometimes called character mode terminals) transmit to the processor all characters, even control characters, as they are entered from the keyboard. This requires that they just be connected by an asynchronous line to the processor since the characters will be generated at irregular intervals.

Terminals in block modes, however, do not transmit characters as they are entered but accumulate them in the terminal's memory. When the transmit key is pressed all the characters in the memory (not including any control

characters) between two bounds are transmitted to the processor.

The actual bounds provided on various terminals vary. A good terminal design will let the user set his own with a default if he doesn't. However some terminals provide fixed bounds using the cursor to delimit one end of the message and the start or end of the current line or the screen to delimit the other. In extreme cases both bounds are fixed. Block mode terminals can be used on either synchronous or asynchronous lines (provided the transmit and receive interface allows this) as the message is sent as a whole and so the characters within that message will be sent at regular intervals.

It is worth considering the relation between the block and conversational modes on the one hand and the full and half duplex modes on the other. Clearly a block full duplex mode is an impossibility (as terminals are currently constructed anyway). The fact the terminal is in full-duplex mode means the character will not be entered into the terminal memory when a key is depressed while the fact that it is in block mode means it will not be transmitted to the processor either!

A summary of the possible mode combinations is as follows: conversational full-duplex mode where, when the key is depressed, the character is sent to the processor but not displayed unless echoed; conversational half-duplex mode where the character is both sent to the processor and displayed when the key is depressed; block half-duplex mode where the character is displayed but not sent to the processor when the key is depressed.

Receive and Transmit modes

Half-duplex lines create problems of control of the line traffic because transmission can be in only one direction at one time. So that the processor at one end of the line and the user at the other must coordinate their activity. This means they must have some method of reserving the line for their sole use or letting the other

end know that the line is required. At the most elementary level control is imposed by providing a "break key" for the user which "breaks" the line. This state can be recognised by the processor even if it is currently transmitting data, causing it to stop and receive some input before resuming transmission.

More sophisticated terminals are able to operate in receive or transmit modes. Unlike the modes discussed earlier, these are temporary modes between which the terminal constantly switches. In receive mode the keyboard is locked out while in transmit mode the processor cannot send to the terminal. These modes are either set and reset by control characters transmitted by the processor or by using a switch provided on the terminal or both.

If the processor has sole control over the terminal modes then a break key must still be provided while if the user has sole control some method must be provided for the processor to signal the fact is has a message to send. A "bleeper" serves this purpose admirably.

Additional Control Functions

This section reviews some extra hardware capabilities possessed by a number of terminals and not discussed previously. These have been left to this point because reference is made to operating modes which had not been introduced till this point. They are discussed under the subheadings scrolling and paging, editing capabilities and protected fields.

Scrolling and Paging

A decision which must always be made when using ADTs is what to do if more information must be displayed than can be fitted on the screen. Terminals attempt to deal with this by providing special control functions which are automatically invoked when the cursor is at the bottom of the screen and a new line of data has to be displayed.

Two slightly different approaches exist. The first is scrolling. Scrolling means every line is moved up one position and the top line is lost from the screen and usually from the terminal's memory. The term derives from the illustration commonly used of a scroll being rolled up

at the top and unrolled at the bottom. Scrolling means that new lines are displayed at the bottom of the screen after those previously received. The second approach, paging, is really wraparound in a vertical direction. Paging moves the cursor to the top of the screen where the next line will be displayed. With paging new lines are displayed at the top of the screen above those previously displayed.

Another hardware facility provided to deal with this problem is the provision of memory which can hold more information than may be displayed on the screen. This may take the form of reserve pages or screenfuls of information which almost instantaneously replace the currently displayed page; alternatively the operator may be able to scroll through the memory in both directions. (This fits the scroll analogy much better since information isn't lost when rolled up but may be rolled down again.) The problem however is merely postponed from the boundaries of the screen to the boundaries of the memory.

Both paging and scrolling on their own are inadequate answers to the problem since at the speed information is displayed, either uncontrolled scrolling or paging is too fast to be read by the operator. Such a solution must be a software one, but it will take advantage of the hardware facilities provided by the terminal.

Editing capability

When a terminal operates in block mode, messages may be composed and corrected before being transmitted. The fact that the characters in the terminal's memory can be changed without the processor being aware of it allows editing of messages. The most basic editing capability is the mere replacement of the character at the cursor position. However it is often desirable to allow more than this, the replacement of one character by two or two by one.

The editing logic which is provided on many terminals takes advantage of the fact that all characters are constantly circulating through a temporary register where the insertion or deletion of extra characters or even lines can be accomplished. Thus, character insertion

where the characters to the right of the cursor are repositioned to allow space for the new character and deletion where the unwanted character is removed without leaving a space are quite simple to implement. Not all the characters to the right of and below the cursor need be moved. This usually depends on the terminal or even different switch-selectable modes available on that terminal. The editing action may apply to the end of the screen, to the end of the memory or, the most common, till (in the case of insertion) two adjacent space characters are found one of which is removed, or (in the case of deletion) one space character is found which is replaced by two.

Line insertion and deletion operate in a similar fashion. Line insertion, where space is made for an additional line at any position on the screen, may be regarded as a generalisation of scrolling which makes space for an additional line at the bottom of the screen only. Again, the line at the top of the screen is lost. Line deletion removes a line and creates a blank line at the bottom of the screen.

These hardware editing capabilities must not be confused with software text editing systems which are, of course, must more powerful.

Protected fields

Protected fields are provided so ADTs may be used for data entry. They allow a "form" to be set up on the screen. Protected fields are set up by the transmission of a "start protection" control character which sets a marker in that position followed by the data to be held in the protected field and finally the "stop protection" control character. Thereafter when the cursor in the course of its normal movement comes across the start of a protected field, it skips through it until it reaches the end of the field and then resumes its normal movement. As a result no data can be put into these fields from the

keyboard because the cursor can never be moved into that area. They are unaffected by nearly all other control actions too. They may not be erased, are unaffected by editing commands and when in block mode and a block of characters is being transmitted any characters falling within a protected area in that block are ignored and not transmitted.

The areas can be marked on the screen by the ability of the screen to differentiate characters by blinking, reversed contrast, a different font or other methods.

What in effect is achieved by protected fields is to make part of the screen immutable. Recognition of this fact has led with some terminals to the setting up of blank protection fields with the information in them not being retained in the terminals but being projected on the screen by a projector which can be used in combination with the terminal.

CHAPTER 2

Interactive Systems which use Alphanumeric Display Terminals

Alphanumeric display terminals are normally used interactively, which means that the terminal is associated with a program executing in the computer to which the terminal is connected. This program receives input from the keyboard of the terminal and generates information which is displayed on the screen in response to the input. Provided that input from the terminal can be processed quickly enough the operator may indulge in conversational intercourse with the program. [2.13]

Three main classes of terminals are used with interactive systems. These are graphics terminals, alphanumeric display terminals and mechanical teletypewriters. These terminals have in common the fact that through them the operator may generate input for the system and also receive output from the system; but they also have marked differences.

The properties of the alphanumeric display terminal which make it suitable for use with an interactive system are that the terminal can display the information it receives as fast as the user can read it and that the display can be selectively updated, essential information being retained while in essential is replaced. The main disadvantages of an ADT are firstly that information can be in alphanumeric form only which renders it unsuitable for all graphic applications; and secondly the lack of hard copy.

It is worth comparing ADTs with the other two major types of interactive terminal. Graphic terminals for the most part can do all that an ADT can do, but they are very expensive, both as regards initial and running costs because a single terminal requires a large amount of processor time and memory. Such considerations render them uneconomic for general use. Cheaper graphics can be provided by storage tubes which maintain the display

by flooding the display area with a beam of electrons. These are much closer in price to ADTs.

Teletypewriters, or teletypes as they are commonly known are cheaper than ADTs although the disparity is being constantly changed in the ADTs favour. This is their main advantage over the ADT. A lessor point is that they provide hard copy of all transactions. This is not necessarily an advantage when a lot of the dialogue is ephemeral and no permanent copy is necessary or even desirable. However completely "soft" systems are very unusual and systems that use ADTs must make provision for hardcopy which means an additional problem for the system designer. When the system incorporates a line printer or hard copy terminal, near to where the ADT is operated the problem can be solved but if the ADT is situated at a distance from the computing system to which it is attached, the problem is a major one. One method of overcoming it is to provide a plug in interface on the ADT through which the memory contents may be written out on a hard copy device (either a teletypewriter or electrostatic printer). If the hard copy device can cope with characters at the same speed as the ADT, then it is really sharing the line, otherwise the ADT may be switched off-line as the memory contents are printed out.

Teletypes lose in comparison with ADTs both in the speed with information is displayed (ADTs can present information faster than a person can read it, teletypes usually present it slower and as the speed of teletypes approaches that of ADTs they lose their price advantage) and in the flexibility with which the information may be presented.

This chapter looks at four major types of interactive systems that currently use ADTs under the headings; Enquiry/Response Systems, Interactive Programming Systems, Data Entry and Computer Aided Instruction.

Enquiry/Response Systems

The largest commercial use of ADTs is in enquiry/response systems. As the name suggest these systems respond with information to the user's enquiry. The aim of such systems is to make accessible to the user information stored internally in a computer system.

The internal files (sometimes known collectively as a data base) can be of considerable size. In order to find information the user must activate a searching program which can be guided to the required information. This "navigation" can involve an extensive dialogue and pages of information must often be displayed as the user comes on the data he desires. This means that ADTs must be used because, in general, teletypewriters cannot cope with the quantities of information satisfactorily.

As well as providing access to information, some systems have routines available which may process the information in the files to provide summaries or statistics. These could be just standard routines that execute in response to a simple command or a facility for the user to construct his own routines. Such programming is performed in a standard interactive programming language or, more commonly, by combining the available routines using programming-like control and assignment statement (This is frequently called a Problem Oriented Language).

Many different kinds of enquiry/response systems have been implemented for various applications. Some examples are systems that allow salesmen to look up an inventory, a policeman or other law-enforcement officers to look up information about stolen goods or criminal records, or managers to look up financial information about their company. [2.1, 2.12, 2.2, 2.11, 2.18, 2.19] The following two subsections deal firstly with the first and most conspicuously successful system under the subheading, airline reservation systems; and secondly with the actual history of a medical information system under the subheading, the medical information system at El Clamino Hospital, California

Airline Reservation Systems

Airline reservation systems were the first enquiry/response systems to be successfully implemented. [2.12] Initially a typical system would use a teletypewriter terminal which had been specially designed for the purpose with a number of unusual features. However a change was later made to ordinary ADTs which in spite of some initial reservation on the part of the operators have proved superior.

With an airline reservation system a trained operator can retrieve from the database all the information necessary for a passenger to make a booking, such as a list of all flights between any two cities on any nominated day with the number of seats currently available and even some of the facilities available on each flight. Then, when the booking is made, sufficient information about the passenger is recorded to allow the booking to be later confirmed, that is, the passenger's name, address, telephone number and any additional information such as if VIP treatment is required. When the time for the flight approaches the booking is confirmed by the operator who specifies the flight and the first two letters of the passengers surname and then gets displayed a list of all passengers whose names begin with those letters (this helps avoid worry about spelling mistakes) with the relevant details about their booking.

One of the reasons why airline reservation systems are among the most succesful of enquiry/response systems is because they enjoy the luxury of a trained operator who is completely familiar with the system. As there is a trained operator an effort is made to reduce line traffic (which is vital in such a terminal-intensive system) by communicating in both directions using mnemonic abbreviations. (For example all airports are referred to by their three letter abbreviation.)

The Medical Information System at El Camino Hospital California

Unlike the airline reservation system described above, other systems have part time and untrained users and so must be more user-oriented. That there are difficulties may be

seen from the history of the Medical Information System at El Camino. [2.15, 2.16]

This system was designed to follow the patient from his admission, record his treatment and produce a bill at the end. The doctor was supposed to use an alphanumeric display terminal to enter his order concerning each patient, for example, for drugs, diet or clinical tests. These orders were processed and lists of instructions were printed out for the nurses to follow. This replaced the old system where a patient's record was held in paper file into which the doctor put his hand written orders and a nurse aide wrote out the instructions for the nurses. The hospital now had a better record of the treatment given the patient and could bill him accordingly.

This system was at first received with hostility by the doctors and nurses because they didn't feel they gained by it. A poll held in the summer of 1972 about six months after the system was first introduced revealed that 66% of the doctors opposed the system and this was borne out by their actions. Doctors, instead of entering their orders, continued to write them by hand and made the nurses enter them instead. Since a lot of the nurses were not trained typists they found this extra duty time-consuming and irksome.

The problem was the system had been originally designed to help the hospital's administration and not to aid the medical and nursing staff. Two thousand changes were made to the system in the first year, a third to remove bugs, a third to improve service to the staff and a third to make the system more robust. As an example of the changed emphasis, extra information was made available to the doctors through the terminal, for example the results of clinical tests, and current medical information such as the available drugs to treat a particular complaint and abstracts of articles in medical literature. Two later polls revealed increasing acceptance of the system. The first in January 73 had 46% of the doctors for and 42% against while the second in July 74 had 61% of the doctors

in favour of the system as were 94% of the nurses. As an example of the increasing acceptance of the system, 70% of the doctors were using the terminals to enter their orders.

The fact that the system became more popular would have in part just resulted from increased familiarity regardless of the improvements, but it still emphasises the fact that the designers of such systems must be careful to orient their systems specifically to the user's needs and abilities.

Interactive Programming Systems

Another major use of interaction is to enable a programmer to develop, run and modify a program online. Such interactive programming systems are often combined with "timesharing" systems so many programs can make use of the computer simultaneously. Interactive programming systems require additional software such as text editors and special compilers or interpreters. Time sharing also requires a powerful operating system. This means such systems usually prove more costly than traditional batch processing. However the programmer is provided with a much more satisfactory service.

Batch processing has the serious disadvantage that delays in turnaround can interfere with a programmer's efficiency. Nothing is more frustrating for a programmer than to have to wait an hour or more for a program to be run only to find that it has not executed because of a trivial syntax error. In an interactive system, the error can be corrected at once and the program can be rerun while with even the quickest batch processing system there will be an irritating delay.

Interactive programming can give to the user of a large computing system the same freedom as the user of a small system who can book the whole computer for his own use. An interactive system can restore to the programmer a feeling of involvement with the computer even

though with current large computing systems he would usually be excluded from the computer operations room.

Interactive programming systems are well-established and, while there are some who disapprove on the grounds that the programmer may be led by such a system not to study his output carefully and thus find all his mistakes, but to make quick fixes, rerun his program and find his mistakes by trial and error, [2.20] they are mostly accepted. Two different kinds of systems exist. The first are general purpose editors which allow the users to use different languages. An example of this kind of editor is described under the sub heading "CANDE, an example of a general purpose editor". The second are special integrated systems based on a simple language. These, which usually consist of an incremental compiler and interpreter with some editing commands, are introduced under the subheading "single language systems". Finally three experimental systems are introduced under the sub headings "the Interactive Programming Support System", "Emily" and "The Automated Computer Science Education System" respectively.

CANDE, an example of a general purpose editor

An example of a general purpose system is Burrough's CANDE [2.6] . (CANDE is an abbreviation of Command AND Edit). CANDE allows modification or inspection of any file stored under the user's name on disc. One special file is designated the workfile and most of the commands affect this.

Some of the things CANDE allows are: sections of the work file may be moved, deleted and whole or pieces of other files can be either inserted or merged according to their sequence numbers; within the work file specific pieces of text may be searched for and replaced; lines in the work file may be changed by the entry of a new line with same sequence number and new lines may be entered if they are preceded by a unique sequence number (a new line is stored between the pair of lines with the nearest larger

and smaller sequence numbers respectively); a file may be passed to a compiler to be compiled and object files may be run.

Also the user has the ability to directly enter many WFL (Work Flow Language, the B6700 job control language) commands.

As can be seen from the above brief description CANDE is a very powerful editor. Also it is very flexible, the Algol, Cobol, Fortran; PL/I, Basic and WFL compilers are available to the user as well as several utility programs. There are limitations imposed by its general purpose nature. For example it hasn't been thought worthwhile to provide special interactive compilers. In fact the source file is passed to the same compiler as used in batch processing systems (Options are set so compilation is aborted after ten errors rather than two hundred as is usual. As well as this, the source file is not listed and error messages are directed to the terminal). However an incremental compiler, as used with special purpose systems, checks the syntax of each line as it is entered and corrections can be made immediately.

As well as this, general purpose editing programs are usually designed to be used with a range of different kinds of interactive terminals. This means usually they are restricted by the properties of teletypewriters because, while ADTs can be used to simulate teletypewriters (and when doing so usually provide a better service because of their greater speed) the reverse is not true. For example editing programs using an ADT should allow modification of a line by cursor movement and overwriting and use of the insert and delete character key, but systems like CANDE do not allow this. (CANDE instead provides the elaborate fix command where the sequence number of the line, followed by the characters to be replaced and their replacement must all be specified).

Single Language Systems

Single Language Systems avoid the difficulties mentioned above. Special incremental compilers can be provided because only one is needed rather than a selection. Interpreters can be provided to let the user monitor the execution of the program while it is being tested. Furthermore, a system can afford to be designed for a single type of terminal.

As an example of this, APL systems [2.10] require a special terminal because of the unique character set the language requires. This character set enables programs to be written with a conciseness of expression which is attractive in an interactive language because it means less typing is required of the programmer when a program is constructed at the terminal. APL systems are tied to their terminals, they cannot be used with ordinary ADTs because of the character set. (Some ADTs have an APL character set available as an option, but this is instead of the usual ASCII characters and thus limits the use of the terminal. This problem of incompatability could be overcome by enlarging the numbers of bits available to represent each character [2.21]).

The nature of the terminal with which the language processing system is intended to be used affects the design of the language. An example of this is the very simple Basic language [2.7] designed for use with teletypewriters. This language is line-oriented and uses the sequence numbers (which must be provided with every line to determine the position of that line in the program) in the branching and conditional branching statements which are the sole means of transferring control within a program.

Basic can be a very useful language and its simplicity allows the provision of an interactive programming system on very small computers. However, the fact that it is line-oriented is a disadvantage with ADTs. Because only a limited amount of the program can be held on the screen at

one time, it is difficult for the user to follow the logic of his program especially if the commands for listing the program are also designed for the teletypewriter (as is frequently the case) and so prevent the user moving backwards and forwards through his program naturally.

The Interactive Programming Support System

With the advent of ADTs, some special interactive programming systems have been designed specifically to make use of them. An example is the Interactive Programming Support System (IPSS) [2.3]. This is based on the Jovial language and allows interactive entry with syntactic checking and editing facilities.

IPSS takes advantage of the fact the screen of an ADT can be selectively updated. The display screen is divided into three autonomous areas. The first is a one line area where messages are placed. The second area is the work area. Here the user enters new program text or his commands to the system. Here also the system can display current program text which the user may modify by use of the built in editing facility of the ADT. The third area, the display area, is used by the system to list part of the current program text. The listing in area 3 is modified as a side effect of editing being performed in area 2 (as well as of course by the action of specific list commands), so the user always has an up-to-date view of the text. Each line is syntactically checked on entry.

The minimum size for the work area is two lines and the minimum size for the display area is twenty lines, so the screen requires at least twenty-three lines (but, as with all interactive programming systems, more would be useful). There are various methods provided for dealing with overflow in the display area including scrolling.

Some conclusions of the implementors of IPSS are worthy of note. They "have found that a display scope for program composition and editing is preferred over a

typewriter-like terminal by most users". [Bratman 68, p1360]. The reasons for this are simple. The user works at all times with an up-to-date copy of his program (either displayed or retrievable by simple commands).

Emily

IPSS was implemented on an IBM 2250 graphics display terminal. This terminal was deliberately chosen because it could be used to simulate different types of ADTs. Another editor implemented using an IBM 2250 is W.J. Hansen's Emily. [2.8, 2.9] This could also use a suitable ADT because the only graphics capability of which use is made is the ability to underline, a feature which is not really necessary.

Emily makes good use of the advantages offered by a display screen. It uses a dialogue to relieve the user of the burden of having to type in his program. Instead Emily uses the menu selection technique to let the user construct his program. Emily employs Backus Naur Form which can be used to define the syntax of a language. A program starts as a single nonterminal and the list of its possible replacements is displayed. The user chooses the successor which will be a string of terminals and nonterminals. Then another nonterminal is selected and a list of its replacements is displayed. Some terminals such as identifiers and constants are entered from the keyboard but the excessive typing that would otherwise be involved in using a language like PL/I (which is the first language supported by Emily) is eliminated.

This method of program construction allows the program to be stored as a tree structure rather than a linear string. This lets the user deal with the program's structural components when editing a program rather than with lines of text. A programmer can look at and display a program in terms of its structure. Indeed, unlike line oriented systems of the Basic ilk where it is difficult to show a

program's underlying structure, by proper use of indentation, the program's structure can be automatically revealed.

The Automated Computer Science Education System

Emily and IPSS are mainly concerned to help the programmer while he is entering his program to ensure the program is well-formed and syntactically correct. The Automated Computer Science Education System (ACSES) [2.22] is concerned to help the programmer, particularly the novice programmer, find errors when running his program. The author states "there is something wrong with batch systems and non-trace systems for small programs which students typically write" [Davis 75,p20] . This is because only an interactive system using interpreters can give the kind of help necessary for a novice programmer who is often bewildered by the cryptic error messages typical of a batch production compiler.

ACSES gives the programmer considerable help. Firstly there is a trace facility. This lets the student see his program executed statement by statement on the plasma display screen at a speed the student can choose. Any any point the student can stop and back up execution to see something again.

As well as this when an execution error occurs the student can employ static and dynamic analysis to trace the cause of the error. Static analysis provides the student with a list of common causes of the error. This list is constantly modified as a result of statistics gathered on the errors being currently made.

Dynamic analysis makes use of the face that execution errors (for example division by 0 or an array having an illegal subscript) are caused by some variable having an incorrect value. Dynamic analysis working backwards one step at time enables the user to see the history that gave that variable its value. At each stage the user is asked if this is the stage where the error was generated. In

this manner it becomes simple for the user to find his error and correct it.

Although experimental systems like IPSS, Emily and ACSES have been implemented successfully, most interactive programming is still done with systems designed for teletypewriters. It is clear however that users much prefer ADTs and so systems in the future must be tailored to take full advantage of the ADTs special features.

Data Entry

Some ADTs have optional interfaces to storage media, like tape cassettes or floppy discs, to allow them to be used for off-line data entry. However most data entry using ADTs is online. In fact both types of system discussed in the two previous sections have data entry aspects. In enquiry/response systems, files are frequently updated online from the terminal and in interactive programming systems new text can be entered from the terminal. This section discusses two other forms of data entry ender the subheadings, "Conventional data entry using ADTs" and "Form filling".

Conventional Data Entry using ADTs

ADTs can be used for conventional data entry. Usually data entry requires "continuous keying" operators who key the data onto some computer accessible medium - for example cards, disc or magnetic tape. ADTs can be used by these operators in a similar fashion provided they have settable tab positions and a programmed duplicate function is provided. The screen in this case is almost superfluous.

The main advantage of using ADTs is the ease of correcting errors. Some checking of the data can be done as it is entered and the operator's attention drawn to possible mistakes. The editing facilities provided by the terminal make correction simple and the screen lets the operator compare visually the text that has been keyed in with the written form. One system that takes advantage

of the ADT's facility is used by a New Zealand firm. [2.5] In this case data is not keyed in at the ADT, but comes from an optical character reader which reads order forms filled in by the delivery men and these are checked and corrected using ADTs.

Generally, continuous keying systems which use ADTs are not part of multipurpose computing systems but have a small dedicated processor which superintends the transfer from key to disc. Later the contents of the disc can be copied to magnetic tape and transferred to the main system.

Form Filling

A more interesting form of data entry that is unique to ADTs is an adaptation of the common process of filling in a form when some information must be recorded. This derives from the feature of some ADTs where protected fields may be set up on the screen. The resulting "form" tells the user what information is required. The fields which the user must complete can be set up to be the correct size for the amount of data required. Each time the user signals the end of a field, the cursor may be positioned at the beginning of the next field. The data can be visually checked by the operator before transmission and then checked by the program and queried if it seems unlikely or incorrect.

In most systems the data is recorded (perhaps on a printed form), transferred to coding sheets, punched by a continuous key operator and then finally entered into the computer. By putting a terminal where the data is generated unnecessary middlemen can be eliminated and terminals can be put almost anywhere - for example the Kansas police carry a small ADT in their car. [2.2] Some examples of the use of ADT forms are systems allowing policemen to fill out accident or arrest reports, salesmen making orders (perhaps while their customers are present), receptionists at a hospital checking in patients or a nurse updating a

patient's record [2.1,2.12].

In fact, under proper control, this technique could be used whenever a form has to be filled out or data recorded, that must be later transcribed into a computer stored file. A minor disadvantage is the lack of hard copy, but provision can be made to have the form immediately printed out (in triplicate if necessary!) and generally such a system proves more satisfactory than conventional systems.

Indeed one author describing the trend to computing systems without card readers or their equivalent but using instead ADTs [2.23] claims "In every installation where CRT data entry replaced key punching, CRT entry was superior - cheaper with fewer errors [Marienthal 75,p63]. The reason is because "with CRTs the function of data entry can be pushed back to the source of data so that the steps of document preparation, edit, keypunch and verify can be combined. The dull grind of punching meaningless numbers is eliminated." [Marienthal 75,p63]

Marienthal supports his case by citing five case histories. Typical of these is a clothing retailer where all ordering, acceptance, dispatching and sale of goods is recorded on ADTs or point of sale terminals at the time it happens, not by data entry staff, but by the ordinary staff in the course of their normal work.

Data entry on ADTs then derives its strength from the ability to control and structure the data entry dialogue so careless errors are not made and misunderstandings about what is required do not occur, the ability to set up "forms" on the screen and to locate the ADT near the source of the data.

Computer Aided Instruction

A pupil who receives "Computer Aided Instruction" (C.A.I.) learns by interacting with a teaching program through a terminal. The technology of the ADT is only a peripheral issue in this new and evolving field but there is no doubt that ADTs are the most suitable terminals.

Unlike teletypewriters, ADTs are fast enough to allow a satisfactory rate of conversation and unlike graphics they are cheap enough to allow the purchase of sufficient terminals so a reasonable number of students can have access to the teaching programs. This section looks at two approaches to CAI under the subheadings "the COPI system" and "the Socratic approach".

The COPI system

Because these teaching programs should be constructed by teachers rather than programmers and should be easy to modify in the light of the users' responses, the emphasis in this field is to provide dialogue generating systems which are simple to use and can be easily changed. Typical of such systems is the Univac COPI system [2.17]. Here the user prepares a sequence of "frames". These consist of the information to be displayed and a list of words to be looked for in the user's response together with the action to be taken if a match is made. This action generally consists of a branch to another frame although a message may be displayed before this happens. The action to be taken when the program cannot match any answer may also be specified.

Menu selection where the user selects his answer from a displayed list of alternatives is a commonly used technique with this kind of system. This is because the teacher may not be able to anticipate all possible answers if the pupil is free to make his own answer rather than choose from predetermined alternatives.

Systems like COPI are easy to use and many worthwhile teaching programs have been constructed using them. COPI and its relatives are designed for the Skinnerian techniques that are usual in CAI. Here the pupil is presented with the information in simple stages, only passing to a new piece when he replies correctly to a question designed to ensure the information is mastered. Incorrect responses cause branches to remedial frames while correct responses are reinforced with branches to new information.

The Socratic approach

Skinnerian techniques are not the only ones possible. Another approach is known variously as the "Socratic", "Inquiry" or "learner-oriented" approach. Here the pupil is posed a problem and must ask for the information which enables him to solve it. If the pupil presents a wrong answer he is led through his deductions until the place where the error was made is found.

The teaching program with such a method is more difficult to write; however one discussion of CAI techniques says that of all forms of CAI the inquiry approach "is the most exciting ... and the most challenging both technically and pedalogically" [Breen 74,p948]

A system like COPI renews the screen completely every time a move is made to a new frame; that is every time the user makes a response. This denies the teacher the full use of the facilities offered by ADTs, especially the ability to update the screen selectively. In a "Socratic" system where the pupil asks for information in unpredictable ways and where it is desirable to retain this information through the dialogue leading to the solution of the problem, such a restriction is undesirable. So while dialogue generating systems must be straight forward and easy to use, this should not necessarily be at the expense of the teacher's ability to make full use of his terminal.

CHAPTER 3

Some Programming Considerations when Using ADTs with Interactive Systems

This chapter discusses some issues involved in the implementation of interactive systems. The first section under the heading "dialogue design" contains some notes on this important aspect of interactive systems mainly indicating the range of possibilities. The second section under the heading "screen management" discusses the problems involved and then reviews strategies employed by some actual systems to overcome them. The third section under the heading "operating modes" describes the implications the different operating modes introduced in the first chapter have for the designer of interactive systems. The final section under the heading "hardware requirements" briefly considers the hardware features desirable for different applications.

Dialogue Design

The design of the form of the conversation between the user and the program is an important part of any interactive system. The two most important considerations for the designer are the type of operator involved and the hardware configuration of the system and the constraints this inevitably imposes. The operator may be a trained and full-time user of the system or at the other extreme an inexperienced and casual user. The types of user should be the most important consideration in the design of the dialogue and the dialogue must cater for them.

Hardware constraints affect the line traffic. In a terminal intensive system the system designer must always work to reduce line traffic. The problem is to avoid achieving this at the expense of the intelligibility of the dialogue.

In this section we briefly survey some of the different kinds of dialogue. A rough distinction can be made between two types of dialogue. In the first control rests with the user. He enters some command, the system executes it, signals the user it has done so and then waits for the next command. This type of dialogue is examined under the subheading "passive systems". In the second case, control rests with the program. After the user has made his presence known, he is queried by the system so as to find what is required. The system then performs the action and queries the user to find out the next action required. We consider this type of dialogue under the subheading "active systems".

Passive systems

Systems where there is a response to the active command of the operator, have a set of commands, the command language, which the operator must learn. The number of commands will usually be limited to a reasonable number; however most of the commands will incorporate parameters which means, although the user is limited to a small number of general actions, a very wide range of specific actions may in fact be performed. For example there may be only one command to achieve listings of files but the incorporation of parameters means the range of things that can be actually displayed on the screen can be very large.

In most cases the user simply enters the command by writing it on the screen. The command could be either full words or mnemonics. Terminal intensive systems with trained operators (for example airline reservation systems) force the operator to use mnemonics for the commands and for the parameters too. Systems with part time operators may prefer to use single word commands and parameters, which can be abbreviated to mnemonic forms as the user becomes more familiar with the system. Full-duplex terminals can even echo the full command when the

user enters the mnemonic.

These approaches assume that the user writes the command on the screen. Another approach is to use function keys. Such an approach is more common with graphic terminals which provide a number of function keys. These can be physically labelled with the commands they represent. ADTs as a rule have very few function keys (although full-duplex mode terminals can allow the use of all keys as function keys) so this approach is not often considered. However there have been systems using ADTs where it has been felt desirable to manufacture a terminal with a special keyboard, each key specifically for the entry of a command. [3.5] There are drawbacks to this approach. The main one is inflexibility due to the fact it is difficult to add commands or incorporate parameters into commands. (The TVEDIT system gets around the last problem by entering the parameters and then using a different control key as a function key to signal the end of the message and indicates the specific command). [3.6] The advantages of the function key approach are that the command is not written on the screen thereby disturbing the screen's contents and the user has a permanent reminder of the available commands written on the keys. However the first advantage can be nullified by good screen management and the second by the provision of the appropriate manual or card.

The other half of the dialogue will depend again on the type of the user and the constraints of the system. The system's response may be cryptic, unintelligible except to the initiated, but this should be if there is only a limited range of responses and if line traffic has to be kept to an absolute minimum. Otherwise the response should be in ordinary language, preferably in full sentences.

Active systems

The alternative to having a system passively responding to the commands of a user is to have the commands actively induced from the user by the system.

One common technique for doing this is "menu selection". Here the user is asked a question and given a "menu" of possible answers among which he must choose. (For example "What is the capital of New Zealand? A Auckland, B Wellington, C Christchurch.") A hypothetical menu is illustrated in fig. 3.1.

If we assume in this unlikely system the user answers 2, the next stage in the dialogue might be as in figure 3.2.

Such a system is unlikely because menu selection is used, firstly where the user is inexperienced and unfamiliar with computing systems and secondly where the range of alternatives is too large for the user to be able to select among them before entering his command. The first situation applies in CAI and the second often applies in enquiry/ response systems where the user is searching for a specific piece of information among a large mass and these are exactly the situations where menu selection can be most usefully employed. The advantages of menu selection are that it needs little or no prior knowledge on the part of the user and that the dialogue can be structured and controlled by the designer of the system. The disadvantages are increased line traffice and the increased screen space required for the dialogue.

Menu selection is not the only technique where the system actively elicits the information it requires from the user. Simple question and answer systems are also common. At one extreme are systems where every question can be answered by either "yes" or "no" or some equally simple response. Such dialogues are designed for systems that are intended for use by the general population. For example medical diagnostic systems where a program attempts to find the specific area of a patient's complaint before the patient actually sees the doctor, have used such a dialogue with some success. [3.5]

At the other extreme are systems which are simply command language systems with some cosmetic cooperation

```
DO YOU WANT TO

   1 BUILD A NEW PROGRAM

   2 EDIT  A CURRENT PROGRAM

   3 LIST A CURRENT PROGRAM

PLEASE INDICATE YOUR CHOICE BY
TYPING THE ASSOCIATED NUMBER HERE -
```

Fig 3.1

```
YOU HAVE THE FOLLOWING PROGRAMS IN YOUR LIBRARY

   1 PROGRAM/ONE

   2 PROGRAM/TWO

   3 PROGRAM/THREE

PLEASE INDICATE THE ONE YOU WISH TO EDIT
BY TYPING THE ASSOCIATED NUMBER HERE -
```

F 3.2

from the system, for example instead of passively waiting for a command the system may display "Please enter your next command".

This of course illustrates the fact that the distinction between active and passive systems is really a false one. They represent two slightly different approaches to dialogue design, both of which can figure in the same system. The most important thing in dialogue design is that the system designer (within the system constraints imposed by the hardware) must try and mould his dialogue to the needs of the user. Since the designer is not omniscient some provision should be made for monitoring the user response to the dialogue to find out those parts the user finds obscure, awkward or confining.

Screen Management

An important aspect of the design of interactive systems which use ADTs is proper use of the resources provided by the screen. There are two problems involved. The first is to reconcile the conflicting requirements for use of the screen. This is looked at under the subheading "Use limitations". The second is the problem caused by the need to display more information than the screen can accommodate. This is examined under the subheading "Size limitations".

Use limitations

In most systems the screen has conflicting demands made on it. For example in an enquiry/response system both the information the user has requested and the dialogue necessary to get it have to be accommodated. In this case it may be possible to use the screen for only one purpose at a time, but where the user is forced to constantly transfer between two different screens of information, there will be frustration caused merely because of the time involved in "blanket" screen renewal. Users should not be forced to retain quantities of

information in their own memories because of a system's limitations. Fortunately the screen of an ADT can be selectively updated. That means the screen can be divided (logically) into autonomous areas. An example of a system where this is done has been introduced previously. As described in chapter 2 IPSS divides the screen into three autonomous areas each with its own distinct purpose. There are difficulties in this approach particularly with hardware control functions which affect the whole screen, but where it can be adopted it should be. At the very least a separate area must be retained for the command dialogue.

Size limitations

One constant problem that arises when writing programs for ADTs is the situation that must inevitably arise when there is more information than can be displayed on the screen. It is hard to think of an application where this problem does not arise.

Two similar approaches are usually taken to this problem to allow the user to move through the text from where he is currently. The first is to fill the screen with as many lines of information as can be accommodated (commonly known as a page), then on depression of the appropriate key to replace it with the next page or screenful. Sometimes the user may also turn back to the previous page. The common alternative to this is scrolling where the information may be "rolled" up the screen, new lines of information appearing at the bottom of the screen and those at the top being lost from the screen. Less commonly provided is the ability to scroll and retrieve lines previously lost off the top of the screen. (This requires the appropriate hardware on the terminal).

These are simple and reasonable solutions to the problem with the scrolling approach being slighty more flexible. But they can be inadequate. Although ADTs display information quickly (especially compared with mechanical devices like teletypewriters) it still takes

a significant time to completely replace a page, (for example it would take six seconds to replace all fourteen hundred and forty characters on a DEC VT05 at the maximum transmission rate) and so if a person has to flip through a large number of pages, it could cause frustration especially if the user wishes to compare information held on two different pages, (for example the text of a subroutine and the place it is called from).

To provide further versatility there are several possible strategies. One is to allow partitioning of the screen allowing a person to move the information which it is necessary to retain to a "save" area and display new information with a reduced page size in the remaining space. A still more common solution is to divide the page into smaller sections and give these a unique name which can be referred to in display commands. Usually this is done by dividing the page into line size sections and attaching to these a unique number. This gives the increased flexibility of letting the display commands refer to individual lines or to groups of lines indicated by a line range.

A different approach is taken by the NLS text editing at the Stanford Research Institute. [3.3] There a hierarchical structure is imposed on the text. Each piece of text is conceived to be on a certain level within this structure. As they say
"3c2b1 The principal manifestation of this hierarchical structure is the breaking up of text into arbitary segments called "statements" each of which bears a number showing its serial location in the text and its "level" in an "outline of the text". [Englebert 68,p398] One reason for this structure is pointed out below.

"3c4b Also in working online at a CRT console, not only is manipulation made much easier and more powerful by the structure, but a user's ability to get about very quickly within his data and to have special "views" of it generated to suit his need are significantly aided by the structure." [Englebert 68,p389] When the text is displayed, the level

to which the display is required and all levels below that depth are omitted. Thus the text is condensed in a natural way, one can read a summary and where more detail is required the user can explore lower levels at that point. Other possibilities provided are the ability to specify a level of truncation so only a truncated version of each statement is displayed, and the ability to specify some "content" for which the system searches, only displaying those statements containing it. Also provided is the previously mentioned ability to retain part of the displayed text in a "save" area. This system certainly provides a useful and flexible way for dealing with quantities of computer stored information, but not all users may care to impose the kind of structure on their text necessary for it to work.

Emily also makes use of the fact that a hierarchical structure is imposed on the program, to enhance screen management. [3.4] In this case the program is stored as a tree structure and to allow display of all the program only certain levels of the tree may be displayed. A whole section of program may be contracted and represented by the symbol of the nonterminal from which it was originally generated. This has the advantage of allowing the user to perceive the whole structure of his program at once, then gives the user the ability to examine particular sections with the necessary detail.

All these various approaches still require a minimum screen area to be effective and work better with more screen area rather than less.

Operating Modes

The alternative operating modes were described in the first chapter. The implications that these operating modes have for the programmer must now be discussed. There are two real alternatives, conversational full-duplex mode and half-duplex block mode, and these provide the subheadings for this section.

Conversational full-duplex mode

Conversational full-duplex mode means that the terminal, connected by a full-duplex line, uses the processor to extend the facilities offered by the terminal. The terminal has of course only the same capabilities hardwired into the control logic but instead of being invoked automatically from the keyboard they are now under the direct control of the processor. This can give to the programmer a great measure of control over what is displayed on the screen. For example, protected fields may be simulated by the program checking if the result of the transmission of a cursor control would move the cursor into the protected area. It may even be desirable to disable all the control keys except those necessary for limited editing of messages to maintain the integrity of what the program has on the screen. For example, if an operator can accidentally invoke the scrolling logic from the keyboard, the whole screen contents may have to be retransmitted.

Of course a price must be paid for this control. The price is increased overheads in processor resources. A copy must be held in the processor's memory of the screen contents and each time a character is received from the terminal, a not insignificant portion of program must be executed by the connected processor. Whether this is worth-while will depend chiefly on the advantages given to the interactive system designer and hence the user by using the conversational full-duplex mode. If the program driving the connected processor only lets the terminal

simulate a block mode terminal (as often happens) then a block mode terminal may as well be used, but the conversational full-duplex mode can have greater potential.

Half-duplex block mode

This mode must be used if the terminals are to share a line or if the terminal is to be connected by a half-duplex line or a synchronous line. There can also be some slight reduction in line traffic and there are certainly reductions in the resources of the connected processor necessary to service the terminal. For simple applications, block mode may be satisfactory, but programs that selectively update the screen may require clever use of protected fields to maintain screen integrity.

Conversational half-duplex mode

Conversational half-duplex mode is not usually a viable alternative because the programmer has the same lack of control as with a block mode terminal in that the user's effect on the screen contents can only be determined after the event and also must pay the same cost of increased processor overheads as with a conversational full-duplex mode terminal - the worst of both worlds. If a conversational mode terminal must be connected with a half-duplex line however this is the only alternative.

Hardware Requirements

For an interactive system to be properly effective it is important that an ADT has the proper hardware features. Unfortunately it is easier to specify features that are necessary than those that may be dispensed with. One or two hardware features as for example protected fields, tabbing, and cursor position read-out, may be simulated by a processor connected to a terminal in conversational full-duplex mode but most features like the basic cursor movements, erasure, scrolling and editing, cannot be thus simulated. (At least not in an acceptable fashion. For instance erasure by transmitting strings of blanks is

usually cumbersome and slow). Hence if they are not available on the terminal the system cannot use them.

Different applications mean different emphases being placed on the various control logic features. If a terminal is going to be used for data entry then more stress will be placed on having protected fields and allowing tab positions to be set by the user as against editing logic. On the other hand if a terminal is to be used with an editing system, the emphasis will be the other way around. This may be a case for making some control logic optional but available if the user requires it.

Other features like screen size and character sets are less variable. At the Stanford Research Institute, two of the criteria for buying terminals for their N.L.S. text editing system are firstly that they "consider 24 lines by 64 characters, a minimum adequate screen size; however 27 lines by 80 characters is much more useful. The most desirable would be a full text page of 66 lines by 80 characters". [Andrews 74,p263]

Secondly "the terminal must display the full ASCII character set, including upper and lower case letters". [Andrews 74,p263] The authors of IPSS concur "An ideal display should be large enough (say 40 lines of 80 characters each) to contain a reasonable sample of data". [Bratman 68,p1359] These criteria are increasingly being met by newer terminals. The trend towards upper and lower case letters is a good one since computer users have always been plagued by inadequate character sets. Screen size is even more important. Recent terminals tend to have twenty lines or more where a few years ago they had ten. However even bigger screens are still desirable. The point at which the increase in cost outweighs the convenience gained by the larger screen is difficult to assess but it hasn't been found yet. If the cost of ADTs continues to fall as dramatically as it has over the period of the last five years, a reasonable screen size should become a reality.

As the screen gets larger so the importance of extra capabilities, like cursor addressing to allow flexibility in the screen management, increases. The screen contents should be disturbed as little as possible in each transaction; blanket screen renewal being most undesirable. While sequences of cursor control characters can be generated to achieve this, cursor addressing is much simpler and quicker. It enables the division of the screen into autonomous areas, something required by most applications. To facilitate this it is helpful if control functions can be made to apply to limited areas. The most important of these is erasure. If the only erasure permitted is erasure of the whole screen, then the terminal is awkward to use. Also line insert logic which can be used to provide generalised forms of scrolling is more useful than just simple scrolling logic which applies to the whole screen.

For the same reasons, that is to allow the screen to be used in a flexible manner, block mode terminals have special requirements. For a start, they should be provided with protected fields, so the programmer has some control over screen contents. Most importantly however it is necessary that messages a user builds can have flexible delimiters, so the user is not forced to overwrite, and hence lose essential information when communicating with the interactive system.

Another possibly desirable innovation is some standardization of control features and, more important, the characters that invoke them.[2.21] Since the ASCII character set (which nearly all terminals use) does not lay down specific characters for specific control functions on these terminals, manufacturers have assigned their own, trying to distort the original intended use of the characters as little as possible. Of course many differences have arisen between terminals both in control functions and the characters that invoke them. Standardization is desirable to allow compatibility provided it is not at the expense of the innovative manufacturer and does not involve a reduction to the "lowest common denominator".

The type of hardware features desirable in ADTs for interactive systems will be further determined only by the design of interactive systems which use exclusively ADTs. In this way areas of need will become apparent. At present many interactive systems are multipurpose - designed for use with both ADTs and teletypewriters. For sales reason manufacturers of ADTs have aided this by trying to maintain compatability between their terminals and teletype. However as ADTs begin to supersede teletypes it will be possible to place more attention on these features that make ADTs unique and much more flexible than the teletype.

CHAPTER 4

Virtual alphanumeric display terminals

In modern computing systems nearly all input and output is performed by routines provided by the operating system. User programs are seldom allowed to control I/O devices directly. This restriction is necessary to allow the resources represented by the I/O devices to be effectively shared among many different processes. However, as a result, the operating system I/O routines can relieve each programmer of the necessity of knowing the intricate details necessary to control each device. Instead the I/O system lets the user frame his I/O commands for a "virtual" device. In this chapter we firstly look at the reasons why virtual ADTs may be considered desirable and then we examine a simple virtual terminal system implemented by the author.

Some reasons for implementing virtual terminals

The reasons just discussed for restricting direct control of I/O devices apply especially to ADTs. In a multi-terminal system, to avoid chaos it is vital that the operating system controls all transactions with the terminals. Also it is convenient to relieve the programmer of the trivial detail involved in communicating with a terminal.

There are some additional reasons for letting the user deal with a software interface rather than the actual terminal. The program may have to be used with different models of ADTs or modified versions of the same terminal. If the programmer can deal with a virtual terminal then he can ignore these differences and allow the software which maps the virtual device onto the actual to make the appropriate changes. There are of course problems involved in this. It can mean that a programmer could be denied full use of the facilities available. If the software interface must make different terminals look identical

then the user can be restricted to dealing with the "lowest common denominator" (that is only using those hardware features that all terminals possess in common). The solution adopted to this problem by the implementors of NLS at the Stanford Research Institute [4.1] was to specify a minimum standard required for terminals bought and if necessary to introduce a hardware (micro-processor) interface between the terminal and processor to make sure the terminal had the capabilities required.

A less drastic solution is to recognise that dealing with the "lowest common denominator" may not necessarily be a hardship in most cases provided the user is allowed to make use of individual features if he is prepared to cope in his program with the extra detail and complexity that would be involved.

The final reason for providing a virtual terminal or software interface is that often the processor connected to the terminal is a special purpose processor within the total computing system, whose sole purpose is communication with terminals. This processor must be provided with a separate program to control it and it would be unreasonable to expect every programmer to provide his own. This program can be used to extend and enhance the capabilities of the terminal hardware especially if the terminal operates in conversational full-duplex mode. Of course once again there is the problem that implementing such a program restricts the individual programmer's options and can deny him full use of the facilities available (unless the terminal is used by only one system in which case the controlling program can be tailored specifically to that system's needs). Two solutions are possible. One is to provide as sophisticated and multi-purpose an interface as possible and hope it will meet the user's requirements. The other is to keep the controlling program as simple as possible, leaving the problem of enhancements to the main program. This solution although inefficient and perhaps - in functions like the echoing of characters - impossible (although

echoing of characters may be precisely the area where different users may have different requirements) does leave the choice to the user.

An example - a software interface for the DEC VT05 ADT

This section describes some of the software which has been written to enable the programmer to control a VT05 from a program in the B6700.

The VT05 operates in conversational half-duplex mode and is linked to the data communications processor (DCP) of the B 6714 computing system by a half-duplex line. As for other terminals, two routines ("requests" as they are known in Burroughs datacom terminology) are executed by the DCP to govern input and output.

The routines are actually dual purpose designed to enable the terminal to be used by the CANDE editing program and for more general purpose use. The CANDE part of the routines makes the VT05 look something like a teletypewriter and inserts essential control characters - this will not be discussed further.

The general purpose parts of the requests used by all other programs are designed to be as simple as possible. The output routine transmits a string of characters one by one to the terminal, inserting necessary delays after certain control characters. The input routine accumulates characters as they are typed in until the ASCII ESC character (whose key is labelled ALT on the VT05 keyboard) is typed in whereupon the whole string of characters is passed to the user program. This has the effect of partially changing the nature of the terminal to a block mode terminal, since it denies the user the chance to intervene after individual characters are received. However since the control characters entered by the user are passed by the DCP to the user program (which would not of course happen with a block mode terminal), it is possible for the user program by maintaining a copy of the

screen contents and simulating the effect of the control characters to update this copy to keep track of the screen contents.

There are two reasons for accumulating characters in the DCP. The first is that it is difficult to control line traffic on the VT05, and this gives some measure of control. The DCP will never enter the output or transmit request while it is still executing the input or receive routine. Hence the user is allowed to key in his whole message before being interrupted. (It is still possible for the user to corrupt the output being displayed by typing while it is being displayed but this is the lessor problem because the user can react to the situation which the DCP cannot).

The other reason is that the user program need execute only one "read" statement for each message string instead of one read statement for each character. Apart from the minor difficulties involved for program organisation, the situation of having a "read" for each character means that at each read statement the program must wait and later be restarted by the operating system. This may cause delays which are most undesirable in conversational systems. However even though the terminal in half-duplex mode, to deny the programmer the chance to intervene after each character is a limitation for the programmer.

For the programmer's convenience, this necessary control software is supplemented by an Algol package designed to make using of the terminal easier. The package works by maintaining two buffers, an input and output buffer, to interface with the VT05. The user, employing the output routines, puts characters into the output buffer and all characters received from the terminal are placed in the input buffer. By scanning the output buffer when its contents are transmitted to the terminal and scanning the input buffer after each "read" from the terminal it is possible to simulate the actions taking place on the screen and thus retain a copy of the screen contents.

The output routines can be briefly described as follows. The PUTSTRING procedure puts a string of characters into the output buffer. This string will be displayed from the current cursor position and "wrapped around" at the end of lines. If it runs off the bottom of the screen, scrolling can take place provided a boolean switch SCROLL is set true. The other major output procedure is the MOVE procedure which puts into the output buffer the control characters necessary to move the cursor to any desired position on the screen. As well as these two vital procedures, several less important procedures are provided. ERASESCREEN puts an erase screen control character into the output buffer, ERASELINE puts an eraseline control character into the buffer and PUTCHAR can put any specified character into the buffer. None of these procedures have any effect on the actual screen contents until the procedure TRANSMIT is called. This sends the contents of the output buffer to the terminal and updates the internally held records of the screen contents.

There are two input routines. ACCEPT puts into any nominated array the string of characters keyed in at the terminal, including the control characters. GETSTRING removes all control characters from the message then transfers it to the nominated array.

This is a fairly low level approach in which the user is confronted with the actualities of what is placed on the screen and where it is to go; however the user can build higher level procedures from these if it is so desired and the screen can be used in any way he wishes.

CHAPTER 5

An example - Constructing Programs with Genisys

Genisys is an interactive programming system which allows programs to be constructed in the General language, using ADTs. It is intended for use by novice programmers although provision will be made for more capable programmers. The system which is implemented on the B6700 computer to which a DEC VT05 ADT is connected, will consist of an editor and an interpreter. This chapter describes a portion of the editor, the routines that supervise program entry.

Genisys borrows heavily from W.J. Hansen's Emily. [5.1] Emily, which was introduced in chapter 2, represents a considerable advance on previous editors that used display screens because the programmer generated his program instead of entering it in the conventional manner. One of the stated reasons for implementing Emily was "to investigate the technique of creating programs from a set of alternatives", [Hansen 71,p684] in other words by the technique of menu selection. The possibility of creating programs in this manner derives from formal methods for defining the syntax of languages. The syntaxes of computer languages may be specified by a formal grammar which defines the set of all possible programs in that language. [5.2] A grammar, described informally, generally consists of a set of nonterminal symbols, a "distinguished" nonterminal from which program derivation starts, a set of terminals and a set of replacement rules. Emily uses the simple Backus Naur (or Backus Normal) form (B.N.F.) grammar. In B.N.F. grammar replacement rules consist of a single nonterminal on the left hand side of the rule and a replacement string of nonterminals and terminals on the right hand side. A derivation tree can be built by starting with the special start nonterminal, selecting one of its replacement strings which forms the next level of the tree and then replacing all the nonterminals in this string and continuing in this fashion until there are no nonterminals

at the leaves of the tree.

Normally a program is entered as a textual string from which the compiler attempts to construct the derivation tree. However, Emily uses the grammar to let the user build the derivation tree explicitly. The user is shown the start symbol and a menu of its alternative replacement strings is displayed. The user selects one and the replacement string takes the start symbol's place on the screen and a menu is displayed for one of the nonterminals in that string. In this way the user can proceed till all the nonterminals are replaced and only a string of program text is left on the screen. Internally however the derivation tree is retained and all subsequent editing of the program must be in terms of this tree structure. This is the process we refer to as "generation" of programs.

Emily proved generation of programs was possible. It also established some of the reasons why generation is desirable. The most obvious advantage is that the programmer can only enter syntactically correct programs. This means the programmer may not have to worry as much about the details of a language and it also means that the parsing step of the compilation is no longer necessary. As far as the practical aspects are concerned there does not seem to be any saving in time to enter the program; however there is less typing involved which may be some incentive for programmers who find typing programs tedious. Also it may prove more satisfying for the user to have the computer reacting constantly at each step rather than typing into an unresponsive terminal.

A more subtle advantage is that generation can lead to topdown construction of programs as the user assembles the control structures and then fills in the details. The program is stored internally as a tree structure rather than forcing the user to modify substrings within the main program string as with conventional editors.

The main problem with Emily is the fact that it requires a graphics terminal. Graphics terminals are expensive to buy and also expensive in terms of the computing resources required to run them. Most computing systems will have at most one of these terminals and time available for its use will be at a premium. For this reason program entry and editing is not likely to be regarded as a particularly suitable application for such a terminal. David Lasker who implemented a similar system to Emily [5.3] using, like Emily, an IBM 2250 graphics terminal sadly concludes that "Unfortunately the cost of the interactive graphics....on the computing facility used to develop our editor makes it financially impractical to use". [Lasker 74,p42] Also of course the scarcity of these terminals means that only a limited number of the users of any computing system would be able to use the editor.

Because of this problem it was decided to adapt the principles of Emily to a system designed for use with ADTs. This system called Genisys (for Generative Interactive System) has one other major difference from Emily. Genisys allows the entry of programs only in the specially designed General language while Emily is a table driven system (that is, syntaxes for any language may be supplied for it and then the user will be able to enter and edit programs in that language). There are several reasons for this difference. Screen renewal on ADTs is slower than on graphics terminals and screen area is considerably less. For these reasons not every language is suitable for interactive use with ADTs. Large general purpose languages like PL/I or Burroughs extended Algol have bulky syntaxes which means that the user has to select from too many alternatives at each stage and to make too many selections before the final program string is arrived at. This means the flexibility of this method of program entry is lost and it becomes tedious quite apart from the fact that the path through the syntax to the final program is not always obvious.

It was decided then to implement a system based on a single language which would be designed to take full advantage of the possibilities offered by generation. As a brief example of the possibilities, one may consider names. Names have been a minor irritant for compiler-writers and they have been forced to introduce various restrictions on their composition. Often they require declaration of all names before their use, a not unreasonable demand for a completed program. However a lot of the problems involved with the parsing of names in a program string disappear with generation. When a name must be used, the current symbol table (or list of names currently known to the system) can be displayed and the user can indicate which name is required. If a new name is to be introduced, the user can enter it at this point and if necessary be questioned as to the type of name intended. Because of the fact that names are represented internally by a pointer into the symbol table, the external representation could be any character string whatsoever. Changing the name when editing is trivial while no problems can arise with reserved words or illegal names. For example a name could consist entirely of digits, no problem could arise because constants and names are differentiated internally. Of course implementers of generative systems may place some restrictions on the character strings that can represent names, but a lot of problems like mis-spelling for instance are avoided.

A perhaps more constructive use of the power of generation is to group and indent lines according to the control structures. Although some programmers do this now when setting out their program, with conventional entry systems it is unreasonable to demand this as a syntactic requirement which must be satisfied before the program will be compiled. However, making the structure of a program explicit in this way is an advantage to the programmer and lets him deal with the program in terms of its component parts when he wishes to modify it or even when he wishes

to merely comprehend and check it. Indentation can be painlessly built into the syntax with generation which it cannot with conventional program entry systems. (Although there is no reason why conventional compilers could not automatically indent program listings, perhaps as an option, if their authors so desired and in fact some do.)

These considerations really motivated the design of the General language. General in its first version (extensions have already been planned) is extremely simple and consists of four control structures and four executable statements. The four control structures are known as groups because the groups and "lines" (which are what the statements are called) which constitute the "subgroups" of any particular group are grouped together and indented accordingly.

The first control structure is the sequential group which is similar to the Algol compound statement in that its constituent groups and lines are executed once in turn; the second is the repetition group which continually executes its subgroups in turn beginning again with the first when the last one has been executed; the third is the conditional group which is similar to the "if...then..." control structure in Algol, that is the subgroups are executed only if the associated condition is true; the last control structure is the choice group which is really a generalised form of the case control structure. Choice groups have a number of conditional groups as subgroups, the first one whose associated condition is true being the one executed. Each group may have an attached comment to explain the purpose of the group and an attached label.

The four lines are assignment, read, print and also the exit line which terminates the group whose label is specified.

The routines which control program entry are similar to a recursive descent parser. Each time a nonterminal must be replaced a new procedure is called for each nonterminal in its replacement. These routines store the

```
2  LOOP:REPEAT
21    READ NEXTNUMBER
22    NEXTNUMBER<0
221       EXIT FROM LOOP
23    SET SUM TO SUM+NEXTNUMBER
24    PRINT "SUM IS"
```

Fig 5.1

```
2  THIS LOOP SUMS POSITIVE NUMBERS
20 LOOP:REPEAT
21    READ NEXTNUMBER
22    TESTS FOR END OF LOOP
220   NEXTNUMBER<0
221       EXIT FROM LOOP
23    SET SUM TO SUM+NEXTNUMBER
3  PRINT "SUM IS "
```

Fig 5.2

appropriate information in the program tree and display the necessary information. However the path through the program is determined not by the contents of the string being parsed but by the way the user answers a series of questions from his terminal. Recursive descent is an elegant and simple method of writing parsers especially on a machine like the B6700 which is designed partly to facilitate procedure calling, recursive calls being treated identically to all others. There are problems involved however in that no procedure can have any knowledge of where it is called. This makes error terminations difficult.

Screen Management

The screen of the VT05 (which has 20 lines each consisting of 72 character positions) is divided into 3 separate areas. The first consists of the two lines at the bottom of the screen. This is the message area where the instructions for the user are displayed and where he types in his reply. It is unfortunate that the same area must be used for these two activities, however there is just not sufficient space to avoid this. Fortunately all the user's responses are generally short. The second area is the main area where the program is displayed. This consists of the first 62 character positions on each of the first eighteen lines. The last area consists of the remaining ten character positions on the first eighteen lines. Here the names known to the system are displayed together with the number that indicates their position in the symbol table.

Each line and group has a line number which indicates the position of its node in the program tree. This is illustrated by the program fragment in fig 5.1. The repetition group is the second subgroup of the main group and all its subgroups are prefaced by the 2. The read line is the first subgroup of the repeat and so is displayed indented three further spaces and given the number 21 which indicates it is the first subgroup of the second subgroup

of the main group. Where a comment is provided, the normal group header information, the "REPEAT" in the case of the repetition group and the condition "NEXTNUMBER < 0" in the case of the conditional group, is accommodated at a special zero'th node.

This line numbering could create problems at the editing stage since each change would involve automatic line renumbering which is most undesirable in conventional linear editors. It may not yet prove to be necessarily a disadvantage with tree-structured systems especially where it is intended the programmer should work mainly with soft copy.

Lines may occupy more than one physical line on the screen. The continuation has no line number and is further indented two positions.

Dialogue Design

B.N.F. is a simple and elegant way of defining the syntax of a programming language, but a system designed for the novice programmer cannot presuppose any knowledge of B.N.F. Because of this Genisys does not display a nonterminal and a list of its replacements, instead the user always adds to the end of the current line if it is incomplete or builds a new line or group immediately under the last completed subgroup in the current group. The user is asked to indicate what he wants next.

Because of the restricted size of the message area, the instructions to the user may be a bit cryptic. Every message is framed as a complete sentence and tries to be as informative as possible in the limited space available. It is not a CAI system, the user must be instructed both in the language and the use of the system. However it would certainly be preferable if each item in the menu could have been placed on a separate line and the implications of each choice spelt out.

The user signals his choice by typing a number. For example the possible responses to the message, "Choose 1 sequential group, 2 repeat group, 3 conditional group or 4 choice group by typing the associated number or type 0 for end of current group", are the numbers 0 to 4.

A possible alternative to this would be a command language approach where the message would be "Choose S sequential group, R repeat group, C conditional group or CH choice group by typing the associated letters or type E for end", where each alternative through out all messages has a unique set of letters associated with it. This may be easier for the user and speed up interactions; however the number approach stops the user racing messages.

Every message displayed expects a response and as far as possible the response to every message causes some change to the program displayed on the screen. Of course this is not possible in every case but it can be frustrating if the user must enter two responses to achieve one object. For example in the first implementation of the routines to control entry of the READ line, the dialogue was as follows. After the first name had been entered, the instruction "If that was the last name type 0 else type 1" was displayed. If the user responded with "1" control passed to the procedure which supervised entry of names and further instructions were displayed. This was found to be irritating and so the two operations, the entry of the names and the signalling of the end of the list of names, being complementary, were combined, the message becoming "Enter the name you require or its number or else type ALT for the end of this line".

There are still areas of the dialogue where the programmer's convenience has interfered with the user's and this is one area where improvements can be made.

Line Entry

Although it is easier for the user to generate lines by selection the experienced user may find it time consuming. It is for this reason that provision is made for the user to enter a whole line at a time. At present the user chooses whether he wishes to generate or enter lines before beginning construction of his whole program although only simple modification would be needed to let the user change from one mode to the other during the process of program generation.

The user can either enter the line as it would appear on the screen (without of course the line number and indentation) or use the abbreviations S for SET, R for READ, E for EXIT FROM and P for PRINT. Also at any point in a line a name may be replaced by its number immediately preceded by the hash character. The line is of course displayed in full however it is entered. If there are errors of syntax in the lines entered then the user is questioned to establish what was intended and the corrections are then made.

At the moment there is an incompatibility between line entry and ordinary generation. In the first case a name may not contain a delimiter (+, -, (,) or blank) while in the second case a name may be a string of any characters (except the question mark which is used internally as a string-end delimiter). Compatibility must eventually be restored, but which way is not clear. Names like "1+1", " " or the null string are probably undesirable but there does not seem to be any reason why names like "sums of squares" or "1st result" or even "X/Y" should not be permitted, apart from the difficulty for the parser.

Using Genisys

To understand the program building phase of Genisys, we will now go through the process of actually building a small program with the help of Genisys.

To do this we respond to the Genisys instructions. When we begin the screen is as in fig 5.3.

If the response is 0 the screen becomes as in fig 5.4. while if the response is 1 the screen becomes as in fig 5.5. The difference, of course, is that with line entry, a line must be typed in while in the other case it is generated.

We now begin to build the main group. After selecting the sequential group we get asked "What does this group do" (as in fig 5.6). This enables the entry of a comment which explains the purpose of the group. After entering the comment the screen is as in fig 5.7.

The label is used in the exit statement which can be used to exit from any level. This labelling system is clumsy although effective and destined to be replaced with a more sophisticated system of terminating a group prematurely which has been introduced in the paper by Zahn. [5.4] We do not label the sequential group.

We can now fill in the subgroups. One of the instructions shown in figs 5.4 and 5.5 is again displayed at the bottom of the screen. We will deal with the building case first. After the system has been signalled that next subgroup is to be an assignment line by typing 1, the screen becomes as in fig 5.8.

The list referred to has at the moment no entries. It consists of all the names known to the program at any point and is displayed on the top right hand side of the screen. Each name has a number associated with it which may be used to refer to that name later in the dialogue. We now enter the name "SUM" and the screen becomes as in fig 5.9.

The name "SUM" is displayed in two places, both in the line itself and in the list previously described. We now build the expression, the value of which will be assigned to the name. In this case the expression is so simple, the constant zero, that it seems irritating that

TYPE 0 IF YOU WANT TO SELECT LINES AND 1 IF YOU WANT TO ENTER THEM

Fig 5.3

TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE

Fig 5.4

TYPE 0 FOR END,ENTER A LINE OR CHOOSE 1 SEQUENCE GROUP,2 REPEAT GROUP 3 CONDITIONAL GROUP OR 4 CHOICE GROUP ELSE TYPE END FOR END BUILD MODE

Fig 5.5

WHAT DOES THIS GROUP DO, TYPE IN ANSWER OR TYPE 0 IF NO ANSWER

Fig 5.6

THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS

IF THIS GROUP IS LABELLED THEN ENTER THE LABEL OR ELSE TYPE 0

Fig 5.7

THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1 SET

SELECT A NAME FROM THE LIST BY TYPING ITS NUMBER OR TYPE IN A NEW NAME

Fig 5.8

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS                    0  SUM
1  SET SUM TO

CHOOSE 0 (,1 NAME,2 CONSTANT,3 +,4 -
```

Fig 5.9

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS                    0  SUM
1  SET SUM TO "0"

TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT
 GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE
```

Fig 5.10

we must first indicate that it is a constant, then enter it and finally signal that the expression is finished, involving three transactions. This is because the system must cater for the most complicated expression. It is however a flaw that could be removed, by treating single item expressions as a special case.

We now have the complete line as displayed in fig 5.10. To get the same effect by line entry, involves only one transaction with the system. The line "SET SUM TO 0" is typed in. It can also be entered by using the abbreviated forms, "S SUM TO 0" of "S SUM 0", to save time.

The next two lines are entered or built in a similar fashion and we have the situation in fig 5.11.

The next subgroup is a repeat group. This must be labelled otherwise we will have a non-terminating loop. It would be possible to prevent the starting of a repeat group unless it or one of its surrounding groups was labelled. After entering the label, comment and first statement which is a read statement, we have the situation in fig 5.12. The repeat group is the fourth subgroup of the main group while the read line is the first subgroup of the fourth subgroup (the repeat group) of the main group, hence their respective numbering.

The next subgroup is a conditional group, that is a group with an associated condition. The condition must be true for the group to be executed. If line-entry obtains then the condition is entered as a whole, in this case as NUMBER = "LAST NUMBER". It may be noted at this point that names already known to the system can for entry purpose be replaced by their number preceded by a hash (" # ") character. That is, the preceding condition could have been entered as " #3 = "LAST NUMBER" ".

Building a condition rather than entering it is the clumsiest part of the system and the most in need of improvement. What is involved is building the expression

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
 1   SET SUM TO "0"
 2   SET COUNT TO "0"
 3   SET SQUARES TO "0"
```

```
0  SUM
1  COUNT
2  SQUARE
```

```
TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT
 GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE
```

Fig 5.11

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
 1   SET SUM TO "0"
 2   SET COUNT TO "0"
 3   SET SQUARES TO "0"
 4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40   LOOP:REPEAT
41       READ NUMBER
```

```
0  SUM
1  COUNT
2  SQUARE
3  NUMBER
```

```
TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT
 GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE
```

Fig 5.12

to the left of the conditional operator, selecting the conditional operator then building the second expression. Building an expression requires a minimum of three transactions so the whole process requires, for the simplest condition, seven transactions. We shall not show this painful process in detail but, at the end, the program as is in fig 5.13.

The only subgroup of this conditional group is an exit line. This exit line is the simplest group to build or enter. When building after the exit line has been selected the screen becomes as in fig 5.14. and "LOOP" is entered. Otherwise the user can enter any of "EXIT FROM LOOP", "EXIT LOOP", "E FROM LOOP" or simplest "E LOOP".

Entry of an exit line automatically terminates the group of which it is a subgroup that is in this particular case the next subgroup built will be a subgroup of the repeat group not the conditional group. This is simply because no subgroup placed after an exit subgroup could possibly be executed. The only disadvantage of this automatic group termination is that there is no visual indication that it has happened and user must be prepared in advance.

We will now look in slightly more detail at the process, to which we have previously referred, of building an expression. We have at this stage after the entry of two more assignment lines the situation as illustrated in fig 5.15, and we wish to complete the line with the expression "SQUARES + (NUMBER x NUMBER)" (the parentheses are there for the purpose of illustration only).

Firstly we choose the name by entering "1" and the next instruction appears on the screen as in fig 5.16.

This restricts names appearing in expressions to these already known to the system. This was intended to prevent the problem of undefined variables but may prove to be irksome. If the system is used as intended for

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
```

```
0   SUM
1   COUNT
2   SQUARE
3   NUMBER
```

```
TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT
 GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE
```

Fig5.13

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
421         EXIT FROM
```

```
0   SUM
1   COUNT
2   SQUARE
3   NUMBER
```

```
ENTER THE LABEL OF THE GROUP FROM WHICH THE EXIT IS TO TAKE PLACE
```

Fig 5.14

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
421        EXIT FROM LOOP
43      SET COUNT TO COUNT+"1"
44      SET SUM TO SUM+NUMBER
45      SET SQUARES TO

                                                    0   SUM
                                                    1   COUNT
                                                    2   SQUARE
                                                    3   NUMBER

CHOOSE 0 (,1 NAME,2 CONSTANT,3 +,4 -
```

Fig 5.15

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
421        EXIT FROM LOOP
43      SET COUNT TO COUNT+"1"
44      SET SUM TO SUM+NUMBER
45      SET SQUARES TO

                                                    0   SUM
                                                    1   COUNT
                                                    2   SQUARE
                                                    3   NUMBER

SELECT THE NAME FROM THE LIST BY TYPING ITS NUMBER
```

Fig 5.16

program evolution rather than for the entry of a pre-written program, and if this is, again as intended, top down program construction, then the user would probably just enter a comment to represent each of the main subgroups and then in arbitary order further refine these groups. Anyone who has written programs in this fashion soon finds they have the need of variables which they should have initialized previously and with the current system problems would arise. With line entry, the problem has been avoided. When the expression is entered it is queried as to whether it was intended or whether it is a mistake. If it is intentional it is entered into the list of names.

After the number "2" has been typed in we have the situation as in fig 5.17.

We can terminate the expression or select an arithmetic operator to continue it. In this case we select the addition operator and once again get the demand that appeared at the bottom of the screen in fig 5.16. This time we select the left parenthesis, it is displayed, and the same demand is repeated. This time after we have filled in the next name we have the situation that appears in fig 5.18. The instruction at the bottom of the screen is similar to that which appeared in fig 5.17 except the expression may no longer be terminated because we have an unbalanced left parenthesis. If at this stage, we selected the right parenthesis then the next instruction from the system would be as in fig 5.17. Of course at this stage we do not select the right parenthesis but the multiplication operator. Then we select the name and finally the right parenthesis giving the complete expression as in fig 5.19.

At this point we signal the end of the expression. We have now finished the repeat group and indeed almost the whole program.

After signalling the end of the accumulating loop we must now arrange to print out the results. The print line is probably the most complex as a list of expressions and format controls must be provided, however as this

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1  SET SUM TO "0"
2  SET COUNT TO "0"
3  SET SQUARES TO "0"
4  THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40 LOOP:REPEAT
41     READ NUMBER
42     WHEN NUMBERS END,TERMINATE LOOP
420    NUMBER="THE END OF THE NUMBERS"
421       EXIT FROM LOOP
43     SET COUNT TO COUNT+"1"
44     SET SUM TO SUM+NUMBER
45     SET SQUARES TO SQUARES
```

```
0  SUM
1  COUNT
2  SQUARE
3  NUMBER
```

```
CHOOSE 1 *,2 /,3 +, 4 - OR TYPE 0 FOR END
```

Fig. 5.17

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1  SET SUM TO "0"
2  SET COUNT TO "0"
3  SET SQUARES TO "0"
4  THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40 LOOP:REPEAT
41     READ NUMBER
42     WHEN NUMBERS END,TERMINATE LOOP
420    NUMBER="THE END OF THE NUMBERS"
421       EXIT FROM LOOP
43     SET COUNT TO COUNT+"1"
44     SET SUM TO SUM+NUMBER
45     SET SQUARES TO SQUARES+(NUMBER
```

```
0  SUM
1  COUNT
2  SQUARE
3  NUMBER
```

```
CHOOSE 0 ),1 *,2 /,3 +,4 -
```

Fig. 5.18

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
421        EXIT FROM LOOP
43      SET COUNT TO COUNT+"1"
44      SET SUM TO SUM+NUMBER
45      SET SQUARES TO SQUARES+(NUMBER*NUMBER

                                                    0   SUM
                                                    1   COUNT
                                                    2   SQUARE
                                                    3   NUMBER

CHOOSE 0 ),1 *,2 /,3 +,4 -
```

Fig. 5.19

```
THIS PROGRAM COMPUTES THE MEAN AND VARIANCE OF NOS
1   SET SUM TO "0"
2   SET COUNT TO "0"
3   SET SQUARES TO "0"
4   THIS LOOP ACCUMULATES THE SUM AND SUM OF SQUARES
40  LOOP:REPEAT
41      READ NUMBER
42      WHEN NUMBERS END,TERMINATE LOOP
420     NUMBER="THE END OF THE NUMBERS"
421        EXIT FROM LOOP
43      SET COUNT TO COUNT+"1"
44      SET SUM TO SUM+NUMBER
45      SET SQUARES TO SQUARES+(NUMBER*NUMBER)
5   PRINT  NEWLINE "THERE ARE " COUNT "NUMBERS"  NEWLINE "MEAN IS -
          "  ,  SUM/COUNT  NEWLINE (SQUARES-(SUM*SUM/COUNT))/( -
          COUNT-"1") " IS THE VARIANCE"

TYPE 0 END,1 ASSIGNMENT,2 EXIT,3 READ,4 PRINT,5 SEQUENCE GROUP,6 REPEAT
 GROUP,7 CONDITIONAL GROUP,8 CHOICE GROUP OR 9 FOR END OF BUILDMODE

                                                    0   SUM
                                                    1   COUNT
                                                    2   SQUARE
                                                    3   NUMBER
```

Fig.5.20

process doesn't cover much new ground we will omit the details. The complete program is displayed in fig 5.20. We select 9 for the end of buildmode and have constructed a simple though clumsy program which has demonstrated the current workings of Genisys.

Summing up Genisys

Genisys is incomplete and any conclusions as to its worth or practicability must wait until a complete system is working. The part of the system described previously is however functioning albeit with faults that could be eliminated. What has been demonstrated is that so far no impenetrable obstacles have yet been found on the path to the completion of such a system and that such a system (unless it is the fond delusion of an author who is too familiar with the system) is reasonably simple and satisfying to use.

APPENDIX 1

Summary of ADT characteristics

Receive/Transmit Interface

Parallel

rates upto $2x10^6$ character/sec

Serial

rates from 110 bits/sec to 19200 (bps)
addressable/nonaddressable
contention and polling line disciplines/no ability to participate in line discipline

Keyboard

Standard teletype keyboard/with separate numeric pad/ with separate control pad

Character Code

ASCII/Other

Control Logic

Standard cursor controls:

forward one space
backward one space
up one line
down one line
"linefeed"
"carriage return"
"home"
Cursor addressing/no cursor addressing
Cursor readout/ no cursor readout
Repeat key/ auto repeat with single key/ no repeat
Wraparound at horizontal boundaries / no wraparound
Wraparound at vertical boundaries/ no such wraparound
Margin at left hand side of screen/ margin at both left and centre of screen

Tabbing:

Tabbing to fixed positions/ Tabbing to programmable positions

Erasure controls:

Erase whole screen/ Erase from cursor to end of screen
Erase whole line/ Erase from cursor to end of line

Scrolling/Paging

Editing Controls:

Insert and delete character
Insert and delete line

Protected fields

Bell

Display Screen

Short-persistance CRT/ Storage CRT/ Plasma screen
Dot matrix characters/ Stroke formed characters
Upper case only/ Upper and Lower case

Memory

Spatial/Nonspatial
Same size as screen/ Extra memory

Operating Capacity modes

Full-duplex/ half-duplex
Conversational/ Block
Receive/ Transmit
Audible receive alarm

Other

Hard copy interface/ No hard copy interface
Tape recorder interface/ No such interface

APPENDIX 2

Survey of ADTs available in New Zealand

Every known supplier of ADTs in New Zealand in June 1974 was asked to supply details of the ADTs they had available. Although an incomplete response was obtained this survey gives some idea of models currently available. Prices are of June 1974.

The terminals described are:

Burroughs Corporation's TD700 and TD800;

Digital Equipment Corporation's VT05 B;

Hewlett Packard Packard's 2615A and 2616A;

International Computers Limited's 7181/2 V.D.U.;

Infoton Incorporated's Vista Basic/Standard/Plus;

Tektronix Incorporated's 4023A;

Research Incorporated's Teleray 3300.

Terminal TD700

Supplier Burroughs Limited,
Government Life Insurance Building,
Customhouse Quay,
Wellington C1 (P.O. Box 399)
contact: Mr B.D.C. Clark

Manufacturer Burroughs Corporation

Receive/Transmit Interface

Asynchronous 75-1800 bits per second
Synchronous 2400-4800 bps
Two wire direct connect (asynchronous half-duplex) up to 9600 bps.
Terminals are addressable
Many line disciplines are available

Character Code ASCII

Keyboard Standard or just numeric keys

Control Logic

Standard cursor controls
Tabbing to fixed positions
Wraparound at all boundaries
Erase whole memory
Insert and delete character
Protected fields
Repeat key
Return key (CR - LF)

Display Screen

Burroughs "Selfscan" plasma screen
5x7 Dot matrix characters
ASCII upper case characters (64 displayable characters)
Screen capacity is 256 characters
Screen configuration 8 lines of 32 characters

Memory

256 characters which can be extended to 1024 characters (3 reserve pages)

Operating Modes

Half-duplex mode only
Block mode only
Receive/Transmit modes set from keyboard
Audible receive alarm

Price

$1600 approximately

Terminal

Supplier as for TD700

Manufacturer as for TD700

Receive/Transmit Interface as for TD700

Character code ASCII

Keyboard as for TD700

Control Logic as for TD700

Display Screen

CRT
5x7 Dot matrix characters
ASCII uppercase characters (lower case available as an option)
Screen capacity 960 characters (may be increased to 1920)
Screen configuration 12 lines of 80 characters on 24x80

Memory 960 or 1920 characters depending on screen capacity

Operating modes as for TD700

Price $2,000 for 960 characters
$2,200 for 1920 characters

Terminal VT05B

Supplier

Digital Equipment N.Z. Ltd.
430 Queen Street,
Auckland
P.O. Box 2471

Manufacturer Digital Equipment Corporation

Receive/Transmit Interface

Asynchronous 110 bps - 2400 bps
Receive and transmit rates need not be identical

Character code ASCII

Control Logic

Standard cursor controls
Cursor addressing
Tabbing to fixed positions
Scrolling
Erase from cursor to end of line
Erase from cursor to end of screen
Bell

Display Screen

CRT
5 x 7 dot matrix characters
ASCII upper case characters
Screen capacity is 1440 characters
Screen configuration is 20 lines of 72 characters

Memory 1440 characters

Operating Modes

Full/Half-duplex
Conversational Mode

Price $2,400 approximately

Terminal 2615A CRT

Supplier

Hewlett Packard (N.Z.) Limited
94-96 Dixon Street,
P.O. Box 9443
Wellington
contact: Mr P. Kelly

Manufacturer Hewlett Packard

Receive/Transmit Interface

Asynchronous 110 - 9600 bps

Character code ASCII

Control Logic

Standard cursor controls except line-feed and carriage return not available
Erase line
Erase screen

Display screen

CRT
5x7 dot matrix
Upper case ASCII characters
Screen capacity 2000 characters
Screen configuration 25 lines of 80 characters

Memory

2000 characters

Operating Modes

Full/Half duplex modes
Conversational mode

Price $2,440 in quantities of 1 to 4
$2,175 in quantities of 4 to 32

Terminal 2616A

Supplier As for 2615A

Receive/Transmit Interface As for 2615A

Character code ASCII

Control Logic

Standard cursor controls
Cursor addressing
Cursor position readout
Scroll and reverse scroll 1 or 25 lines
Tabbing and back-tabbing to programmable positions
Erase from cursor to end of line
Erase from cursor to end of screen
Insert and delete character
Insert and delete line
Protected fields
Bell

Display Screen

CRT
5x7 dot matrix characters
ASCII upper and lower case characters
Screen capacity 2000 characters
Screen configuration is 25 lines of 80 characters

Memory

The memory can hold 256 lines of characters, that is 20480 characters

Operating Modes

Half/Full duplex modes
Block/ Conversational modes

Price

$3,870 in quantities 1-3
$3,526 in quantities 4-32

Other

Hard copy interface

Terminal 7181/2 V.D.U.

Supplier International Computers (NZ) Ltd.,
Securities House,
The Terrace,
Wellington 1.
P.O. Box 394
contact: Mr R.J. Payne

Manufacturer International Computers Limited

Receive/Transmit Interface

Parallel

10^6 bps that is 125,000 character/sec

Serial

600 bps - 4800 bps
on asynchronous or synchronous lines
The terminal is addressable

Character code ASCII

Control Logic

Standard cursor controls plus ability to move one space on each of the diagonals
Cursor addressing
Tabbing to fixed positions
Erase screen
Erase from cursor to end of screen
Erase from cursor to end of line
Scrolling
Insert and delete character
Insert and delete line
Protected fields
Bell

Display Screen

CRT
5x7 dot matrix characters
Upper and lower case ASCII characters
Screen capacity 2000 characters
Screen configuration 25 lines of 80 characters

Memory

2000 characters

Operating Modes

Block mode (Messages have flexible delimiters)
Receive/Transmit modes set by both processor and from keyboard

Price

$130 a month rental which includes maintenance

Other

Hard copy interface
Badge reader security device

Terminal Vista Series. Basic, Standard and Plus

Supplier

Geo. W. Wilton & Co.Ltd.,
410 Hutt Road,
Lower Hutt,
P.O. Box 3-556
Lower Hutt
contact: Mr P.M. Glenn

Manufacturer Infoton Inc.

Receive/Transmit Interface

Parallel

15,000 characters/sec

Serial

110-9600 bps on asynchronous or synchronous lines
Vista Plus terminals are addressable, and can participate in many line disciplines

Character Code ASCII

Control Logic

Standard cursor controls
Cursor addressing
Erase screen
Insert and delete characters
Insert and delete lines
Protected fields
Bell

Display Screen

CRT
5x7 dot matrix character code
Upper case characters only on Basic, upper and lower case characters on Standard and Plus
Screen capacity 1600 characters
Screen configuration is 20 lines of 80 characters

Memory

1600 characters

Operating Modes

Half/Full duplex modes
Block/Conversation modes

Price

Range from $2,000 - $4,500 depending on options

Other

Hard copy interface
Tape cassette interface

Terminal 4023A

Supplier

W.L.K. McLean Ltd.,
103-5 Felton Mathew Avenue,
Glen Innes,
Auckland.
P.O. Box 3097
contact: Mr B.G. Wisniewski

Manufacturer Tektronix Inc.

Receive/Transmit Interface

Serial 110 bps to 9600 bps on asynchronous or synchronous lines

Character code ASCII

Control Logic

Standard cursor controls
Cursor addressing
Cursor position readout
Tabbing and back-tabbing to fixed positions
Erase from cursor to end of screen
Erase page
Insert and delete character
Insert and delete line
Protected fields

Display Screen

CRT
5x7 matrix dot characters
ASCII upper and lower case characters
Screen capacity 1920 characters
Screen configuration is 24 lines of 80 characters

Memory

1920 characters

Operating Modes

Full/Half duplex modes
Block/Conversational modes

Price

$2,550 excluding sales tax, $3,060 with sales tax

Other

Hard copy interface

Terminal Teleray 3300

Supplier

I.D.L. Utilities Ltd.,
P.O. Box 37-060,
Parnell,
Auckland,
contact: Mr Trevor Eagle

Manufacturer Research Inc.

Receive/Transmit Interface

Asynchronous 110-2400 bps

Character code ASCII

Control Logic

Teletype compatible cursor controls only
Repeat key
Scrolling
Bell
Return (CR - LF) key

Display Screen

CRT
5x7 Dot matrix characters
ASCII upper case characters
Screen capacity 960, 1728 or 1920 characters as options
Screen configuration: 24 lines of 72 characters,
24 lines of 80 characters

Memory

as for screen capacity

Operating Modes

Conversational Mode only
Full-duplex/Half-duplex

Price

$AUS 1355 + duty and sales tax
(price for 1920 character capacity)

Other

Hard copy interface

APPENDIX 3

The Digital Equipment Corporation VT05 Alphanumeric Display Terminal

This is a reasonably inexpensive and simple terminal. It must be connected by an asynchronous data communications line to a processor. (In this case the data communications processor (DCP) of the B6700). It operates exclusively in conversational mode and though it can operate in both half and full duplex modes, it has been designed basically to operate in full-duplex mode because there is no way provided of coordinating line traffic when in half-duplex mode. It transmits and receives using ASCII 67 code.

The screen capacity is 1440 characters arranged in 20 lines of 72 and this is also the size of the memory. The displayable character set is 63 characters, consisting of the upper-case alphabet (lower-case letters being displayed as upper-case), the digits 0-9 and twenty special characters - they are displayed using a 5x7 matrix. It can transmit the full ASCII set of 128 characters and receive 106 of that set, the rest being treated as the null character. The transmit and receive rates range from 110 bits per second (11 characters a second) to 2400 bits per second (240 characters a second). They need not be identical. The keyboard is a standard teletypewriter keyboard which can generate all 128 ASCII characters when extended by the Shift and Control keys. Also provided for convenience are eleven control keys and two function keys. The cursor is automatically moved forward one space when a new character is displayed but there is no wraparound. The control characters allow the cursor to be moved forward or backward one space, up or down one line, back to the start of a line (carriage return) or to the home position on the screen. Fixed tabbing is provided. This moves the cursor to the position to the right that is the nearest multiple of 8 (counting from 1) except for the last eight positions where the cursor is only advanced one place. Cursor

addressing is generated by the transmission of the cursor addressing character and two normal displayable characters which are not displayed but interpreted - the first giving the line position and the second the position on that line. Five null characters must follow the line address characters to allow the control logic time to effect the move.

When a linefeed character is received in the bottom line, scrolling takes place. The eraseline and erasescreen control characters allow erasure of portions of the screen. The eraseline control character erases all characters from the cursor to the end of the line and erase screen does this and also erases all lines below the cursor. A bell rings when the cursor passes the 64th position of the line or when the ASCII bell control character is received by the control logic.

APPENDIX 4

General's Syntax

Program → subgroup

subgroup → line group

group → repetition group | sequential group | option group | choice group

sequential group → optional comment optional label list of subgroups

repetition group → optional comment repetition list of subgroups

option group → optional comment condition list of subgroups

choice group → optional comment choice list of option groups

repetition → optional label REPEAT

condition → optional label expression relation expression

choice → optional label CHOICE

line → assignment | read | print | exit

assignment → SET variable TO expression

read → READ list of variables

print → PRINT list of expression control

exit → EXIT FROM label

relation → < | > | =

control → NEW LINE | NEWPAGE

expression → primary optional list of operator primary

primary → optional { + | - } { constant | variable | (expression) }

operator → + | - | x | /

APPENDIX 5

Description of Buildprogram

Introduction

This is an informal description of the Burrough's extended Algol procedure BUILDPROGRAM and some of its more important sub-procedures. The calling of sub-procedures is governed by the syntax of General and the users response to a displayed request.

Firstly the input/output procedures which Buildprogram uses, are briefly described then Buildprogram and the procedures it invokes are described pseudo-algorithmically. Names and strings in capital letters denote actual names and strings from the program. All procedures in capitals are somewhere described.

Input/Output primitives for Buildprogram

The screen is divided into three areas and each is referenced by a different routine. QUESTION outputs a message to the instruction area. DISPLAYNAME puts a name into a name table on the right of the screen. DISPLAY is the routine that puts a string into the next position on the screen. This position is adjusted each time by DISPLAY and also by the routines INDENT which signals the next line is to be indented three further spaces and NEWLINE which signals the next string is to be placed on a new line. These routines also adjust the line number which is automatically displayed, if the string to be displayed is the first on that particular line.

GETANDSTORECOMMENT asks the user
"WHAT DOES THIS GROUP DO, TYPE IN ANSWER OR TYPE 0 IF NO ANSWER"
The comment is recorded and if no comment is entered this is indicated to the calling routine.

GETANDSTORELABEL is similar. It tells the user
"IF THIS GROUP IS LABELLED THEN ENTER THE LABEL OR ELSE TYPE 0".
The label is recorded and if there isn't one the calling routine is appraised of the fact.

```
BUILDPROGRAM
     Prepare screen.
     Prepare structure which holds program in internal form.
     Ask the user to
"TYPE 0 IF YOU WANT TO SELECT LINES AND 1 IF YOU WANT TO
ENTER THEM"
     Set the Boolean switch LINEENTRY according to the users
     response.
     Enter program node into structure, call BUILDSUBNODES and
     finally move structure to permanent position.
END BUILDPROGRAM


BUILDSUBNODES
     (Buildsubnodes assembles the subgroups of any group)
     Repeat the whole sequence until the user signals the
     end of the current group or an exit line is entered
     or the boolean flag TERMINATEBUILDMODE is set true.
     If LINEENTRY is true
     then
     ask the user to
"TYPE Ø FOR END, ENTER A LINE OR CHOOSE 1 SEQUENCE GROUP,
2 REPEAT GROUP, 3 CONDITIONAL GROUP, OR 4 CHOICE GROUP ELSE
TYPE END FOR ENDOFBUILDMODE"
     and according the users response
     exit from the current incarnation of BUILDSUBNODES,
     call GETANDPROCESSLINE,
     call BUILDSEQ,
     call BUILDREP,
     call BUILDOPT,
     call BUILDCHOICE,
     or set TERMINATEBUILDMODE to true
     (terminate buildmode is set false in buildprogram
     and when set true, will terminate all incarnations
     of buildsubnodes)
```

```
     ELSE if lineentry is false
     ask the user to
"TYPE 0 END, 1 ASSIGNMENT, 2 EXIT, 3 READ, 4 PRINT,
5 SEQUENTIAL GROUP, 6 REPEAT GROUP, 7 CONDITIONAL GROUP,
8 CHOICE GROUP OR 9 FOR END OF BUILDMODE"
     and according to the user response exit from the
     current incarnation of buildsubnodes

     call BUILDASSIGN,
     call BUILDEXIT (and then exit from the current incarnation
                     of buildsubnodes)
     call BUILDREAD,
     call BUILDPRINT,
     call BUILDSEQ,
     call BUILDREP,
     call BUILDOPT,
     call BUILDCHOICE,
     or set TERMINATEBUILDMODE true

END BUILDSUBNODES.

BUILDSEQ

     (Buildseq builds a sequential group)

     Call GETANDSTORECOMMENT (assuming there is one if
     there isn't a boolean switch NOCOMMENT will be set true).

     If there is a comment call INDENT which sets the
     variables which store the position where the next
     string is to be displayed so it will be displayed
     indented three further positions.

     Store the sequential group node (with any comment
     attached) in the program structure.

     Call GETANDSTØRELABEL (assuming there is one)

     Attach the label node to the sequential group node
     (if there is no label, the label node will be set
     to show this).

     Call BUILDSUBNODES

     Attach the subnodes to sequential group node

END BUILDSEQ
```

BUILDREP

(Buildrep builds a repeat group)

Call GETANDSTORECOMMENT.

Store the repeat group node in the program structure (with any comment attached)

Call GETANDSTORELABEL

Attach label node

Display "REPEAT" in the next available position on the screen

Call NEWLINE which adjusts the variables that hold the positions where the next string will be displayed on the screen so it will be on the next line.

Call INDENT so that the next line will be indented 3 positions (for further explanation of NEWLINE and INDENT see the section on input/output procedures)

Call BUILDSUBNODES

Attach and store the subnodes

END BUILDREP

BUILDOPT

(Buildopt builds a conditional or option subgroup)

Call GETANDSTORECOMMENT

Create option group node (with any comment)

Call GETANDSTORELABEL

If LINEENTRY is true
then
call CHECKCOND
ELSE if linenetry is false
then call BUILDCOND

Call NEWLINE then INDENT

Call BUILDSUBNODES
Attach and store subnodes

END BUILDOPT

BUILDCHOICE

(Buildchoice builds a choice group)

GETANDSTORECOMMENT,
Create choice group node

GETANDSTORELABEL,
Attach label node.

Call NEWLINE then call INDENT

Repeat the following section until the user signals there are no more option subgroups or TERMINATEBUILDMODE is set true

Call BUILDOPT, then
ask the user

"IS THIS THE LAST OPTION IS THIS CHOICE GROUP TYPE 0 FOR YES 1 FOR NO"

and according to the users response exit or continue the loop
Attach and store subnodes

END BUILDCHOICE

BUILDASSIGN

(Buildassign builds an assignment statement)

Display "SET" on the screen
Call BUILDNAME,

Display "TO".

Call BUILDEXP,

Create a new node and attach the name node and expression node to it.

Create an assignment line node and attach the previous node to it.

END BUILDASSIGN.

BUILDEXIT

(Buildexit builds an exit statement)

Display "EXIT FROM".

Ask the user to

"ENTER THE LABEL OF THE GROUP FROM WHICH THE EXIT IS TO TAKE PLACE"

Check the label the user enters against those in the label table and if it is incorrect tell the user

"INCORRECT LABEL, TRY AGAIN"

then repeat the process with the new label.

Display the label.

Create an exit line node with the label attached.

END BUILDEXIT

BUILDREAD

(Buildread builds a read statement)

Display "READ"

Call BUILDNAME

Askthe user

"IF THAT WAS THE LAST NAME PRESS ALT, OTHERWISE ENTER NEXT NAME OR ITS NUMBER"

then if the user has entered a name or its number, display it and if it is a new name, enter it in the name table and display it on the screen in the name table and then go back and repeat the sequence

otherwise if the user has indicated the list of names has finished end the sequence.

Create a read line node and attach the list of names.

END BUILDREAD.

BUILDPRINT

(Buildprint builds a print statement)

Display "PRINT"

Ask the user to

"SELECT EITHER 1 CONTROL, 2 EXPRESSION BY TYPING 1 OR 2, OR TYPE 0 FOR END"

and according to the user responses:
ask the user to choose between newpage and newline, record the choice and repeat the sequence;

Call BUILDEXP then repeat the sequence;

or terminate the sequence.

Create and store a print node with the list of expressions and controls attached.

END BUILDPRINT.

BUILDEXP

(Buildexp builds an expression. There are two phases to buildexp. In the first phase the expression is entered and the individual components are stored in a stack, in the second phase the expression is ordered and stored in a tree structure according to the priority of the operators and restring indicated by brackets).

Repeat the entry sequence until user ends expression

Ask the user to
"CHOOSE 0 (, 1 NAME, 2 CONSTANT, 3 +, 4 - ?"

According to the users response:

display "(" and then, incrementing the variable LEFTBRACKETS by 1,

call BUILDEXP (leftbrackets is a global variable, set to 0 outside the first call of Buildexp and accumulating it's value through all incarnations of Buildexp invoked when an expression is being built) and finally display ")";

call BUILDNAME,

call buildconst (which merely gets the users to enter the constant and records it);

or if the user has chosen either + or - (this means the user has chosen a signed primary) the user is questioned as to the name or constant which constitute the rest of the primary.

Next according to the value of leftbrackets the user is asked:

(if leftbrackets is 0) to

"CHOOSE 1 x, 2/, 3+, 4- OR TYPE 0 FOR END";

of (if leftbrackets is greater than 0) to

"CHOOSE 0), 1x, 2/, 3+, 4-".

In either case if the user chooses 0 the loop is terminated, the expression is unstacked, ordered as a tree structure and stored; otherwise the operator chosen is stacked and the loop is repeated.

END BUILDEXP

BUILDNAME

Buildname asks the user to

"SELECT A NAME FROM THE LIST BY TYPING ITS NUMBER OR TYPE IN A NEW NAME"

This name is checked, and if it is a new name it is entered in the name table and displayed in the name table on the screen.

The name is then displayed in the next position available on the screen.

END BUILDNAME.

GETANDPROCESSLINE

(Getandprocessline takes the line that the user has entered and parses, displays and stores it. If the line is syntactically incorrect, the user is questioned as what he really intended and the corrected line is displayed and stored. The procedures called by getandprocessline will not be described since they resemble so closely those already described with the difference that the flow of control is directed by the next token in the line the user entered rather than by the user's response).

Call GETNEXTOKEN

If token's first letter is"S"then call getasgn;
if token's first letter is"E"then call getexit;
if token's first letter is "R" then call getread;
if token's first letter is "P" then call getprint;
of if the token does not begin with any of those four letters, the user has made a mistake so the user must be told

"A LINE MUST BEGIN WITH SET, EXIT, READ OR PRINT. REENTER THE LINE"

and after the line is reentered the process is gone through again.

END GETANDPROCESSLINE

CHECKCOND

(Checkcond parses, displays and stores the condition)

The user is asked to

"ENTER THE CONDITION THAT DETERMINES WHETHER THE GROUP IS EXECUTED"

This condition is split in two expressions by the conditional operator (< > =) in the middle.

If there is no condition operator in the string then the user is told

"EXPRESSION MUST CONTAIN < , > OR =. REENTER THE CONDITION"

and the sequence is repeated, otherwise the first expression is parsed and displayed, the conditional operator is displayed and the second expression is parsed and displayed.

Finally the conditional expression is stored.

END CHECKCOND.

BIBLIOGRAPHY and REFERENCES

Chapter 1

Asten, K.J. (1973)

Data communications for business information systems. New York. 359p

Blazek, O. (1975)

A microprocessor-scanned keyboard. Hewlett Packard Journal Vol.26 No.10: p6-10.

Burroughs Corporation (1970)

B9353 Input and Display system reference information. City of Industry, Calif. 77p.

Burroughs Corporation (1973)

TD 700/800 Equipment. Detroit. 120p.

Davis, S. (1969)

Computer Data Displays. Englewood Cliffs, N.J. 379p.

Digital Equipment Corporation (1973)

VT05 Alphanumeric Display Terminal Reference Manual. Maynard, Mass. 30p.

Doub, J.A. (1975)

Cost-Effective, Reliable CRT terminal is first of a family Hewlett Packard Journal. Vol.26 No.10: p2-5

G.T.E. Information Systems (1971)

Ultronic Videomaster 7000 Series. Mount Laurel, New Jersey, 8p.

Infoton Incorporated (1973)

Maintenance instructions VISTAR. Document No. 02325, Burlington, Mass. 47p.

Infoton Incorporated (1973)

VISTAR technical users manual. Document No. 2326, Burlington, Mass. 33p.

Infoton Incorporated (1973)

VISTAR/GT technical users manual. Document No. 02421, Burlington, Mass. 23p.

International Computers Limited (1973)

The 7180 Series alphanumeric visual display unit. 23p.

Lane, A.B. (1975)

A functionally modular logic system for a C.R.T. terminal. Hewlett-Packard Journal Vol.26 No.10: p6-10

McLaughlin,R.A. (1973)

Alphanumeric display terminal survey. Datamation 19, 11 (Nov); p71-92.

Omron Systems Inc. (1973)

8025 CRT terminal. Sunnyvale, Calif. 4p.

Plews,D.E. (1971)

Displays: technology and potential. In N.C.C. The new technologies, Maidenhead, England.

Roy,J. (1975)

A high-resolution roster scan display. Hewlett-Packard Journal Vol.26 No.10: p11-15.

Sherr,S. (1970)

Fundamentals of Display System Design. New York, 480p.

T.E.C. (1973)

Series 400 Data Screen terminals. Tucson,Arizona. 6p.

T.E.C. (1973)

Mini-Tec data-screen terminal. Tucson,Arizona, 8p.

Tektronix Computer Products (1974)

1974 International Catalog. Beaverton, Oregon. 48p.

Waitman,T.F. (1975)

Firmware for a microprocessor-controlled CRT terminal. Hewlett-Packard Journal: p16-19.

Whiting,J and Newman,S. (1975)

Microcprocessors in CRT terminals. p41-46 In AFIPS Conf.Proc. Vol.44, Montvale,N.J.

Chapter 2

Ref.no.

2.1 Asten,K.J. op.cit

2.2 Bockel,M. (1972)

Kansas City Police telecommunication network system. p67-92 In Online 72, International conference on interactive computing, Uxbridge.

2.3 Bratman,H.; Martin,H.G. and Perstein,E.C. (1968)

Program composition and editing with an on-line display. p1349-1360. In AFIPS Conf.Proc. Vol.33 Part 2 (FJCC) Montvale,N.J.

2.4 Breen,C.P. (1974)

Computer-assisted instruction in industry. p947-951. In AFIPS Conf.Proc. Vol.43 (NCC) Montvale,N.J.

2.5 Butler,B.G. (1972)

Implementation of an interactive system in a commercial environment, p213-238. In NZCS Third National Conf.Proc. Vol. 1.

2.6 Burroughs Corporation (1972)

B6700/B7700 Command and Edit (CANDE) Language Information Manual. Form No. 5000318. City of Industry, Calif. 105p.

2.7 Digital Equipment Corporation (1970

PDP-11 Basic Programming Manual. Form No. DEC-11-XBPMA-B-D. Maynard,Mass. 90p.

2.8 Hansen,W.J. (1971)

Graphic editing of structured text, p681-700. In Parslow,R.D. ed. Advanced Computer Graphics, Plenum Press, London.

2.9 Hansen,W.J. (1971)

User engineering principles for interactive systems, p.523-532. In AFIPS Conf.Proc. (FJCC) Vol. 39, Montvale,N.S.

2.10 I.B.M. Corporation (1971)

Apl/1130 primer. 3rd.ed. White plains, New York. 193p.

2.11 Huyuck,P.H. (1973)

C.A.I. techniques for information retrieval. Datamation Nov. p.91-92

2.12 Martin,J. (1973)

Design of man-computer dialogues. Englewood Cliffs, N.J. 559p.

2.13 Miller,R.B. (1968)

Response times in man-computer conversational transactions, p267-277. In AFIPS Conf.Proc. Vol.33 (FJCC) part 1, Montvale,N.J.

2.14 Van Dam,A. and Rice,D.E. (1971)

Online Text Editing: A survey. Computer Surveys Vol.3 No. 3.

2.15 Yasaki,E.J. (1973)

What will they zap at El Camino? The CRT on the two year old MIS. Datamation, Oct., p142-146.

2.16 Yasaki,E.J. (1974)

What most of the doctors ordered. Datamation, Sept. p138-139.

2.17 Sperry Rand (1974)

COPI - A programming language for education. Document EDS 101-B. 8p.

2.18 Sutcliffe,W. and Denne,T. (1972)

Online enquiry the easy way, p149-170. In NZCS Third National Conf.Proc. Vol. 1.

2.19 Yourdon,E. (1972)

Design of online computer systems. Englewood Cliffs, N.J. 608p.

2.20 Hamming,R.W. (1970)

Computers and computing in the 70s, p5-16. In NZCS 2nd.Nat.Conf.Proc. Vol. 4.

2.21 Fletcher,J.G. (1974)

Characters in a dialogue. Datamation Oct. 9

2.22 Davis,A.M. (1975)

An interactive analysis system for execution-time errors, UIUCDCS-R-75-695. University of Illinois at Urbana-Champaign, Urbana, Ill.

Chapter 3

Ref.no.

3.1 Andrews (1974)

Line-processor - A device for amplification of display terminal capabilities for text manipulation, p257-265. In AFIPS Conf.Proc. (NCC) Vol 43, Montvale,N.J.

3.2 Bratman, (1968) op cit

3.3 Engelbart,D.C. and English,W.K. (1968)
A research center for augmenting human intellect, p395-410. In AFIPS Conf.Proc. (FJCC) Vol.33 Part 1, Montvale,N.J.

3.4 Hansen, (1971) op cit

3.5 Martin, op cit

3.6 McCarthy,J; Daw,B; Feldman,G and Allen,J. (1967)
THOR - a display based time sharing system, p623-633. In AFIPS Conf.Proc. (SJCC) Vol 30. Montvale,N.J.

Chapter 4

Burroughs Corporation (1971)

B6700 Network Definition Language Information Manual. Form 5000078. City of Industry, Calif. 108p.

Burroughs Corporation (1973)
B6700/B7700 Data communication (datacom) Functional Description. Form no. 5000060. City of Industry, Calif. 58p.

Doran,R.W. (1974)
The Lord of the Disks, Report No.16. Massey University Computer Unit, Palmerston North 18p.

Hoare,C.A.R. and Perrott eds (1973)
Operating System Techniques, London

Irby,C.H. (1974)
Display techniques for interactive text manipulation. p247-255. In AFIPS Conf.Proc. (NCC) Vol 43. Montvale,N.J.

Chapter 5

ref.no.

5.1 Hansen, (1971) op.cit

5.2 Thompson,I (1975)
A comparative study of formalisms for programming language definition: Thesis. Massey University, Palmerston North. 122p.

5.3 Lasker,D.M. (1974)
An Investigation of a New Method of Constructing Software, Technical Report CSRG-38. Computer Systems Research Group, University of Toronto.

5.4 Zahn,C.T. (1974)
A control statement for natural top-down structured programming. In Lecture Notes in Computer Science Vol 19. Paris.