14 Pp.

Text Entry, Analysis and Correction Help

Assisting the Disabled Computer User with Data Entry

Guy Hirson

Dissertation submitted to

The University of Cape Town

in fulfilment of the requirements for the degree of

Master of Science in Medicine in Biomedical Engineering

Cape Town

August 1990

The University of Cape Town has been given the right to reproduce this thesis in whole or in part. Copyright is held by the author.

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

I, Guy Hirson, hereby declare that the work contained in this thesis is my own (except where acknowledgements indicate otherwise). This thesis is being submitted for the degree of Master of Science in Medicine in Biomedical Engineering at the University of Cape Town and has not been submitted before for any degree or examination at any other university.

Signed: . Signed ..... Date: 13 August 1990

Abstract:

It was suggested several decades ago that computers would be the single biggest step forward in integrating people with physical disabilities into "normal" society. At that stage, much work was done in writing software and designing hardware that allowed computer operators **with** disabilities to use packages effectively, in certain cases as efficiently as people **without** disabilities.

Since those days, judging by the lack of references on this subject the interest in dealing with disabled people has waned. It is only very recently that the spotlight has been focused on these potentially very productive persons. Unfortunately, the backlog is large and most existing applications software offers little or no support for users with disabilities.

In this thesis, I have examined some of the hardware and software limitations of current desktop computer technology, focusing on the IBM PC and compatibles. I have also written a computer program that attempts to relieve some of the difficulties faced by a limited number of disabled users. In evaluating the results, I considered it important to relate the ensuing data with the real problems faced by a far wider spectrum of users than I attempted to cater for with the program and to suggest ways in which software products could be made to have wider applicability in the future.

I, Guy Hirson, hereby declare that the work contained in this thesis is my own (except where acknowledgements indicate otherwise). This thesis is being submitted for the degree of Master of Science in Medicine in Biomedical Engineering at the University of Cape Town and has not been submitted before for any degree or examination at any other university.

Signed: ..... Date: 13 August 1990

Dedication

This thesis is dedicated to Annalu Waller and the other disabled people of the world who are trying to make a go of it against all the odds that we non-disabled people have thrown in their way. May they all succeed as admirably as she has done.

I give thanks to my supervisors, Lyn Hanmer and Dr David Boonzaier for their assistance, guidance and support.

My friend, Ilse von Willich, deserves a mention, for her encouragement and insistence provided the staying power to complete this dissertation.

"I strongly recommend that interfaces be designed by teams consisting of programmers, psychologists, and task-specialists, all working together."

D.A.Norman - First USA-Japan conference on Human-Computer interaction. Honolulu, Hawaii - 18-20 August 1980

Table of Contents

Appendices

List of figures

List of abbreviations

Please note that certain abbreviations are used in the following text for brevity:

C.P.	=	Cerebral Palsy
PC	=	Personal Computer (particularly IBMs and compatibles)
MS-DOS[1]	=	MicroSoft-Disk Operating System
TEACH	=	Text Entry, Analysis and Correction Help

---

[1] MS-DOS is a registered trademark of MicroSoft

APPLE][(mentioned later) is a registered trademark of Apple Computer, Inc.

Chapter 1 - Introduction

1.1 Background

This dissertation was brought about after viewing a disabled computer user attempting to enter text via the keyboard of an Apple][micro-computer. The user in question suffers from cerebral palsy (C.P.) and has difficulty co-ordinating her movements. She also evidences the hyperextension of joints characteristic of this pathology.

She is also a competent computer user (having completed a degree in Computer Science) and should be classed as a "good" typist considering her disability. However, it came to my attention that a major problem seemed to arise from mis-striking the small keys on the keyboard and then attempting to correct the resulting error.

It occurred to me to write a "smart" interface program that would make editing of errors simpler by dividing the keyboard into larger target areas. After tinkering with the idea and writing a simple program, I was persuaded to expand the project into a thesis with the hopes that research and application of the findings in the area might result in a package that would be useful to this user and other persons having the same sort of disability.

This dissertation is the culmination of the work that was undertaken. It looks at the existing methods of communicating with computers and their advantages/disadvantages and how to improve them. It also considers the statistical methods used to determine when the user should be offered help and alternatives to these. The program offers the user alternative steps of action to take if inconsistencies are detected and eases the task of editing errors. An evaluation was carried out to determine the good and bad effects of such an interface program.

1.2 Computer architecture and I/O

It is always interesting to look back on the history of a particular item's development and criticize it with the aid of hindsight. It is also of value to learn from the mistakes made and to use this knowledge to moderate the path of future development. Finally, it is an experience to see how interwoven many disciplines become in this development.

Computers, far from being an exception, probably fit into this mould better than most other advancements of modern times. Besides breaking the developments of computers up into obvious categories such as programming, electronics and electrical engineering, mechanical engineering (think of the sophistication of modern disk drives), and so on, there are many subtle disciplines that creep in.

For example, consider the input/output (I/O) structures. Excluding the obvious backgrounds of electronics and mechanics, it can be seen that ergonomics and information ergonomics play an important part. These are, respectively, the consideration of the user-machine interface in physical terms, i.e. aesthetics, ease of use, feel of key movement, crispness of display, etc. and the consideration of the user-software interface, covering menu vs. command language driven programs, display generation, etc.

Radiology may be considered as playing a part if the radiation and ionisation effects of cathode ray tubes (the common TV screen) are taken into account. In many cases, optics are beginning to play an important role as light signals are used to communicate between peripheral equipment and the main computer. The terminal also requires cooling with its attendant fans, air flows and filtration units, hence air flow analysis and thermal dissipation studies are required. Linguistics is receiving more attention due to recent interest in voice communications and analysis of text for various purposes. The list could be extended almost indefinitely.

While it is exciting to see such synergy, it is not always as complete as it seems. Taking ergonomics and information ergonomics into special consideration, major flaws begin to arise. Perhaps a lack of involvement of true "ergonomists"

is the cause. Perhaps more specialised persons should have been called in to give advice. Keyboards are an historic jumble of ideas and many other developments do not take all users into account, especially when the "disabled" or unusual user is considered. I shall inspect the keyboard and its alternatives more closely in the next chapter and discuss some of their strengths and weaknesses.

1.3 What makes a user "disabled"?

By definition, in this context, a disabled user must be someone who wishes to enter data to the computer but is handicapped by physical or mental difficulties. These make it awkward or impossible to accomplish such data entry by conventional means.

Examples of such users range from mentally retarded people, through psychomotor disordered users to those who are paralysed and have no use of their limbs at all. They are people who may be unable to understand the concept of entering data via standard I/O devices or people who are not physically capable of doing so. They may be people who are able to use the normal methods but only at substantially reduced speeds. (These users are discussed further in section 2.5.)

Such users form a considerable proportion of the population. Typical figures are (Dept of Health 1987):

Cause of disability	Proportion of population
Cerebral Palsy	0.25%
Autism	0.02%
Intellectually handicapped	3.00%
Physically disabled	1.63%

1.4 Available literature

When I wrote the first demonstration program with this smart interface, I started to look for literature that might describe the concept more fully. Alternatively, other, similar uses of keyboards were a likely target. Although my work was original in concept, I was certain that the idea must have been examined in some form or other before. However, it is an unfortunate truism that very little attention has been paid to the disabled computer user to date. Consequently, not much literature has been written on the subject of assisting users with disabilities to enter information and utilise computers in general and none suggests that anyone has tried to provide a more usable keyboard using the standard QWERTY keyboard that is delivered with personal computers. In the past,

alternatives to the keyboard and different layouts of the existing keyboards have been tried, but not specifically for users with disabilities.

However, some sources, such as PC Computing (July 1989) and Byte (August 1983), have attempted to draw some attention to the problems of computer users with disabilities. Such articles usually prove limited in their scope, giving an overview of some of the equipment available and a couple of case histories. It is to be hoped that the new focus on people with disabilities (with a recent "Year of the Disabled" having gone by) will encourage more attention to these issues in future publications.

Writing this dissertation required that I provide some historical detail on the existing keyboards and describe attempts to improve the usability of these keyboards. I found information for the chapter dealing with these matters in the references from Lemmons (1982) and the 2 papers from Gilad and Pollatschek (1986) amongst others. Bailey (1982) also describes keyboard developments as one of the many facets of the human-machine interface dealt with in this wide-ranging text.

The TEACH (Text Entry, Analysis and Correction Help) program presented in this thesis attempts to modify its data and interface to suit the circumstances prevailing at the time an error is detected. Literature that I did find

useful in this area included the unpublished paper by Benyon where the concept of self adaptive user interfaces is discussed. While the techniques used in Benyon's MONITOR are not directly applicable, the paper provided much food for thought. The same could be said about the papers by Innocent (1982), and the unpublished papers by Macaulay and Norman and by Maskery (both undated).

Publications by Crystal (1985), Butler (1985) and Marcus (1982) proved valuable in preparing a discussion on linguistics and the applicability of this technology to text analysis, while Bailey (1982) again proved useful when it came to statistical analysis.

The publication from McDougall, Knysh, et al. (1988) is a valuable document that makes the first significant attempt to draw guidelines for software developers who wish their programs to be usable by disabled and non-disabled people alike. I didn't see this text until after TEACH had largely been written, but I did attempt to encompass some of the concepts in the most recent upgrade to the program. This text is important to mention in any case, as I would recommend it to anyone writing any software intended for use by human operators. If it had been published some time ago and utilised by the authors of most of today's applications software, this thesis would probably not have been relevant.

Finally, mention must be made of the publication by Angermeyer, Jaeger, et al. (1989). This publication does not cover any single part of the thesis in its applicability, but was invaluable in producing recent advances to the software written. Also, Bailey (1982) has already been mentioned a couple of times, but it must be stated that this book provided data for many of the chapters, inspiration for much of the work performed and a good guideline to ergonomic design in general. The thesis by Annalu Waller, the person who originally inspired this project, provided the texts used in the evaluations (Waller, 1989).

1.5 What lies ahead?

Having laid the foundations of the problem before us, consider the components of the problem in more detail. Bear it in mind that the problems of user-to-computer data transfer are to be examined **as well as** how to determine where errors have occurred so that something can be done about it. It would also be useful to know more than simply that a mistake was made. Reasonably "smart" assistance is the objective.

In the next chapters the keyboard, its alternatives and their shortcomings are examined in some detail. I will then consider various means available for analysing

incoming text so that inconsistencies can be detected (not necessarily errors - this must still be determined).

Since this latter part becomes a major function of the software (overcoming the inherent problems of the keyboard is more innovation than effort!) quite some time and paper is spent on looking at the options available and where the basic starting point is. I will proceed to examine the upgrading required to go beyond this basic stage and look ahead to see where the end of the road leads.

Then the practical implementation of these ideas (both the keyboard improvements and the analysis) is demonstrated as well as some of the other software employed to make the final product more useful.

Finally, I will explain the means used to evaluate the software, the results and the criticism of them, both subjectively and objectively. Also, I shall look at the suggestions that ensue and how to put them to some use in future software developments.

Chapter 2 - Human/computer interfaces and users

2.1 Interface developments

Examining the history of the keyboard and alternative I/O structures that have been developed up until now, evidence of many of the inter-disciplinary involvements described in chapter 1 can still be seen. Of primary interest at present is the ergonomics of the data input structures, but it is important to consider the more mundane aspects of these developments in order to see why basic ergonomics principles were or were not followed.

Early computers were provided with data via complex arrangements of switches which were totally incomprehensible to anyone but the hardware expert who was probably involved in the design of the machine in the first place. Software as we know it today was an almost non-existent artform and all data was comprised of on's and off's (or 0s and 1s), forming native machine code. As the computer matured, it became clear that there was a need for higher level computer languages that were written in a more English-like form (computers originate in the UK and the USA). Standard text input devices were required.

This was not too serious a problem as the difficulty had already surfaced when typewriters were developed in the 1890's and a solution had arisen then. In fact, a number

of solutions had been proposed. In any case, the Universal keyboard (now known as the QWERTY keyboard) was adopted as a standard despite the existence of other, superior designs (Lemmons, 1982). The acknowledged reason for the acceptance of this layout is that it reduced the likelihood of type hammers jamming. When adjacent hammers were used in rapid succession, the heads collided at the platen and stuck there. The layout of the QWERTY keyboard reduced this problem by placing the hammers such that adjacent letters formed unlikely character pairs. The keytops were then labelled accordingly. An empirical but effective approach.

The adoption of an alphanumeric keyboard standard solved the problem of text entry and hardware design proceeded. Keyboards have a number of disadvantages however. They tend to be particularly confusing and slow for the newcomer and other 'reluctant' typists (more about this later). They are also inferior to other data input means for particular applications. For example, the joystick provides a far more practical games interface in many cases. The mouse is useful for cursor manipulation in graphics environments, as are graphics tablets, light pens, etc.

2.2 State of the art

For those who are not familiar with the terms mouse, graphics tablet and others, a range of the most common computer input devices are describe below.

2.2.1 The mouse

The mouse is a hand-sized device that has a rubber ball, or other means of detecting movement, on its underside which is set up so that movement of the mouse across a flat surface is translated into signals that convey position and movement data to the computer. It usually has two or three buttons on top for additional input and is connected to the computer via a cord, giving it the characteristic mouse shape. Its operation is such that a large movement will cause the cursor to move correspondingly large distances across the screen while fine movement allows for final placement of the cursor. The buttons usually indicate that the current cursor position has some specific significance.

2.2.2 The graphics tablet

This is a slab that lies on a flat surface (usually in front of the screen). A special pen or crosshair pointer is used to indicate some position on the pad.

Typically, there is a one-to- one relationship between the tablet and the screen itself so that pointing to any particular location on the screen is simple. Frequently, border regions of the tablet are devoted to menu type command/function areas. Overlays are generally used to define these locations.

2.2.3 The light pen

The light pen is a pen-like device that is connected to the computer by a cord. It has a light sensitive detector inside it and is interconnected to the display control hardware so that pointing it to a location on the screen informs the computer of the location of the pen on the screen. A button serves the same purpose as those on the mouse.

2.2.4 The touch screen

Similar to the light pen in effect, an interleaved array of light beams passing across the surface of the screen indicate a location when the beams are broken by a finger or other thin object.

2.2.5 The trackball

Identical in basic construction to most mouses, the trackball consists of a ball whose movement is sensed.

It is rolled by sliding the palm of the hand across its surface. Buttons are also found on trackballs.

2.2.6 The joystick

Much like the joystick of the aeroplane from which its name is derived, this device consists of an upright pole which is moved around by the operator to indicate movement in a particular direction. Buttons are available.

2.2.7 Vocal input

Not a device as such, voice input of data is receiving much research attention as an alternative method of communicating with machines. The field may be considered to be in its infancy still as very few practical offshoots have resulted thusfar. Essentially what is implied here is a speech detector and analyser front end to the standard input peripheral.

There are other devices used for data entry, but this summary covers the more common ones and will serve to provide the basis of general discussion on the topic.

2.3 Ergonomics and keyboard shortcomings

Besides the work carried out in developing alternative data entry channels, some amount of study has been devoted to modernising the standard typewriter keyboard to suit modern requirements. This research has taken account of the keyboard layout and the mechanics of the keys themselves.

The supposed advantages of the QWERTY keyboard expounded in section 2.1 were fine for mechanical typewriters, but the reasoning certainly does not apply to electric typewriters or computer terminals. In fact, the only likely excuse for its retention is the ground base of typists already familiar with the existing keyboards. Revised layouts have been examined, some with startling results. Dvorak determined that it would be logically more expedient to place the vowels at the home positions of the left hand and the five most commonly used consonants under the right hand fingertips (Lemmons, 1982 and Gilad & Pollatschek, 1986). This layout had the additional advantage of retaining the same, standard, physical construction as the popular QWERTY configuration. Somewhat surprisingly, studies have failed to show any advantages in typing speed over the QWERTY layout and in some cases the QWERTY layout proved superior! This finding was made even though the users had long periods of acclimatization to the keyboard they used. Some DVORAK keyboard users had never even used QWERTY keyboards at all. Adler and others have proposed completely designed

keyboard architectures that have keys contoured to fit the shape of the hands, generally curving concavely on a flat surface so that the fingertips rest naturally on them, or placed on non-planar surfaces to achieve the same effect.

Other advances in keyboard design have been the developments of tactile feedback to give the keys a 'live' feel, contoured keytops to make them more 'hittable' and audible feedback (clicking). Less superficial alterations have increased longevity, reliability and made the keyboard less prone to damage from dust, liquids, etc. These have, no doubt, benefited all users.

2.4 Common problem areas

Section 2.1 implies that, while standard keyboards are unusable by some disabled people, there are other devices which are not. In section 2.2 I discussed a number of common alternatives to the keyboard. With the exception of the last, they all suffer from one common problem. If someone is incapable of using a simple keyboard, the fine movements required to control those alternative devices become an insurmountable task. These channels are fine for healthy users, but not suited to most users with disabilities. This statement is not universally true, but frequently applies. Some of the devices may vary in their sensitivity and their applicability for use by people with disabilities may change in accordance with the sensitivity.

For example, fairly rough control is sufficient to make a wheel-chair move reasonable smoothly under joystick guidance. However, for intricate data entry, such sensitivity is not suitable, nor is it generally available.

As an example of the problem stated here, the C.P. user of chapter 1 was questioned regarding her reluctance to use a mouse, even when it was almost imperative for efficient operation of the graphics package she was using. The reasons given were that she was unable to place the cursor exactly where she wished without absolute control over the smallest distances traversed on the screen (i.e. one pixel). This was not possible with a mouse since her hand moved the device in too jerky a manner, rendering it useless to her. The same argument applies to the other devices described.

However, other devices have appeared to circumvent these problems. The alternatives range from keyboards with very large keys (and similarly structured graphics tablets) or keyboards with configurable size keys (the "Unicorn Expanded Keyboard" (PC Computing, 1989)), through simple switches (activated by elbows, the head, etc.) to blow/suck tubes. Again, these devices offer some serious problems. One is that they tend to be slow. Frequently the user must interface to the computer via scanning software that automatically steps through the options and the user indicates when the desired option is reached. Scanning

takes time since a reasonable delay must be included to allow the user to react before selecting the indicated option. Other specialised keyboards have also been developed such as the 2DOF based on a VIC-20 computer (Johnson, 1986).

These alternatives also tend to be costly since they are not produced in large quantities. There is a limited market and the producers are usually specialists. Overheads are high and this is all carried through to the end user. Since the average disabled user is already financially disadvantaged by having unusual medical expenses, inadequate earnings and the need to purchase many utilitarian items required because of their particular impairment as well as the purchase of the computer, these additional expenses are often beyond their reach. Informal discussions with staff at the Independent Living Centre (Johannesburg) and South African National Council for the Disabled have yielded limited information about the expenses incurred. Physical figures are not available from them or any published source but a rough figure of R500 per month is estimated purely for bladder and bowel maintenance in the case of a paraplegic/quadriplegic. Costs must include adaptations to living quarters to accommodate the disability, added transport costs (potentially including the modification of a vehicle), purchase of special equipment (e.g. wheelchairs, etc.) and medical expenses for treatment, medication, etc. Bear in mind that many Medical

Aid Societies will not provide for already disabled people, or they extract a high premium in these cases. Many would regard computer equipment as a luxury. For some disabled people, it becomes a necessity if they are to be reasonably integrated into mainstream society.

2.5 Cerebral palsy and similar disabilities

In the case of this particular study, only a small section of the disabled user spectrum is considered. This particular group of "reluctant" keyboard users comprises persons who suffer from cerebral palsy, multiple sclerosis and other users afflicted with similar motor disorders that prevent them from using a conventional keyboard at "normal" rates. ("Reluctance" here may be related to an inability to use it properly, regardless of the amount of effort put in.)

The psychomotor disorders referred to above stem from a partial or complete lack of communication between the psychomotor area of the brain (precentral gyrus) and the individual muscles used to execute a particular motor function. This communications failure could stem from total or partial lesions within the brain itself (such as for C.P.), within the spinal column or in the peripheral nervous system. Alternatively, other alterations in the ability of nervous tissue to perform its communicative function could be responsible - e.g. the demyelinisation of

axons that results in M.S. The path that such communication must follow includes the cerebellum, the postcentral gyrus (somatosensory area), the basal ganglia, the associative motor cortex, the Tectospinal tract of the spine, the Pyramidal tract, various portions of the brainstem and small centres in the subcortical areas of the cerebrum as well as the peripheral nervous system. Since the nervous system operates using feedback, the sensory tracts of the spine and brain also have some relevance here.

A failure to communicate accurately, or process data correctly in any of these areas produces some psychomotor disorder. For example, a transection of the Tecto-spinal tract produces spastic vestibulo-visual orientation (movement of upper torso, head and eyes in orientation to sound and balance stimuli). Dysfunction of the cerebellum results in incoordination, tremors and loss of balance. Cerebral palsy (spasticity) results from lesion(s) in the brain inhibiting this communication, presenting symptoms as described above and the characteristic hyper-extension of joints.

The problem faced here is that the user is able to hit roughly the right area (where the target key is) but not always with sufficient accuracy. The result is frequent mis-striking of the intended key, usually with consequent entry of an adjacent key's character. Further problems can

result when an attempt is made to correct the error. The backspace (or some other editing key) must be struck to remove the mistake and the correct key selection attempted once again. If the error is not detected immediately, more than one keystroke is usually required to remove the error.

Note that the above description cannot include all cerebral palsied or users with disabilities since there is a requirement that the user be able to use the keyboard, albeit not too proficiently.

Chapter 3 - Computer analysis of text

3.1 Introduction

Various methods are used to analyse text streams in order to perform any of a number of functions on the text. The most common application is probably for checking spelling, but translation, grammar checking, compilation of dictionaries, statistical analysis of style, and many other uses are envisaged or are already employed.

Some are fairly simple to implement in a computer program, others require far more effort and time. Research in the area is still very much in progress and many of the techniques are only experimental. I shall describe the most widely used of the current techniques and describe the basic pitfalls of each as well as the benefits.

3.2 Statistical analysis

The program written for this thesis utilises statistical analysis of text in order to locate probable errors in the incoming text stream from the user. The approach taken is to derive a table of probabilities for all letter combinations in English (covering only 2 or 3 letter combinations) and use this to flag low probability letter sets coming from the user. If a pair or triplet of letters (called digrams and trigrams respectively) fall below a

specified acceptability level probabilistically, then the user is alerted and may choose to accept assistance or reject it.

Since this is the approach used for the thesis, I have gone into great detail describing the advantages and disadvantages of the technique in the next two chapters. However, the most obvious advantage is that it is a fairly simple technique to implement in code. The major problems are that it is not a very reliable technique for finding errors and is very memory-intensive.

3.3 Lexical analysis

Lexical analysis is the examination of text by breaking the input stream into sequences of separate words and analysing each word in isolation. Complete lexical analysis then requires finding the root of the word (its lexeme or lemma) and classing the word in terms of its modification of the basic lemma (Butler, 1985). For example, the word **their** would be based on the lexeme **my** which has many other modifications in regular use: his, hers, mine, etc. In practice, many programs that utilise lexical analysis do not perform lemmatization. They are interested in the word itself and variations of it, but not of the entire class of words that have similar semantic use.

The problem of defining where a word starts or ends is a major problem experienced with this technique. Obvious word terminators are spaces (although there are many characters in the computer that may be classed as a space, e.g. the TAB character) and most punctuation. However, some punctuation presents a problem:

Hyphenation may be regarded as a special case - but is a hyphenated word regarded as a single word without a hyphen, or must the hyphen be included as a fundamental part of the word? Moreover, some hyphenated words are clearly two separate words conjoined by the hyphenation.

Words may include an apostrophe. Should these words be expanded if the apostrophe is a contraction of some other word? Is the genitive case (e.g. the **doctor's** scalpel) a single word or do we drop the trailing part of the word including the apostrophe?

Lexical methods are widely used for spelling checker programs. A dictionary is used (usually compacted) and incoming words are checked when they are complete. The problems are mainly associated with the dictionary. It must perforce be large if it is to check the spelling of most of the words in the general language, it is limited (usually) to one language and it requires the whole word to be entered before it can look for an error. Its advantages are that it can usually correct your entire word for you

(if it is in the dictionary) and that dictionary compaction allows fairly large dictionaries to be kept on disk without degrading the response time of the system to subjectively unacceptable levels.

A good example of this technique being applied in practice is the program Turbo Lightning (Borland International).

3.4 Linguistics

Linguistics is a large topic (with many tomes written on the subject) and is worth a little more elucidation at this stage.

First of all, it must be pointed out that, in terms of the definition of linguistics, the techniques described above for the evaluation of text could be considered to fall under the general umbrella of *linguistics*. However, my use of the term employs a slightly tighter definition than merely "the study of language". I consider the term "language" to mean *natural language*. This means that statistical analysis and lexical evaluation do not fall into this category as these techniques can be applied to any data based on iconic information (letters of the alphabet are iconic), including numerical data. Syntax, or grammar, is of no importance to these techniques. My interpretation of *linguistics* is the opposite of this : the

syntax and semantics of the analysed text data are of major relevance.

Linguistic analysis then encompasses phonology (breaking the stream up into smallest distinct sounds (*phonemes*)), syntactic analysis (*grammar*) and *semantics* (examination of the meaning of the portion of text) (Crystal, 1985 and Butler, 1985).

3.4.1 Phonology

While it may seem that the use of phonics or pronounceability would not really be applicable to evaluation of the written word, nothing could be further from the truth! Recent developments in this field have encouraged many programmers in the word-processing field to consider the use of phonology for spelling checking purposes. The words often supplied by recent spelling checkers seem to bear little similarity to the original, supposedly mis-typed word. Their appearance often seems vastly different, let alone the letters used. However, it is often less obvious (we are reading, not speaking, remember) that the pronunciation of the words is frequently similar (this is also referred to as *homophony*). (This technique utilises the so-called *Soundex* code to check for similar sounding portions of text (Krammerer, 1988).)

3.4.2 Syntax

There is at least one grammar checking program available that apparently analyses syntax and/or grammar and looks for inconsistencies. At this stage, the methods used for this type of evaluation of text are still a little incomplete. The program in question, Grammatik III, still settles for some of the most basic errors in the English language (e.g. confusion of single/plural noun/verb mixes, etc.) although it has generally received good reviews. Hence, syntactic analysis of text is clearly a reality and offers fairly usable operation already. It is obviously better at finding grammatical errors than typographical ones, though. Ultimately, its use for checking typing while entry of text is in progress is limited as it needs complete parts of speech to operate. Evaluation of the program Grammatik III supports this argument (Baran, 1988).

3.4.3 Semantics

When applying the concepts of semantics, we step firmly into the realms of *Artificial Intelligence* (AI). There is a lot of work in progress that attempts to have computers "understand" the meaning of textual data. The programs that try to achieve this must emulate human thinking to a certain extent as

much of the meaning in the spoken or written word cannot be explained using simple rules (Marcus, 1982). Information is often contextual and may involve informal usage of vocabulary (e.g. slang). It usually depends on mood (which can frequently be detected from the semantics of the data, on the other hand) and many other vagaries of the originator of the text. If humans are frequently confused when confronted with unfamiliar phrases, how can we expect computers to be otherwise.

3.5 Closing comments on text analysis

It is clear that there are many techniques to be applied to text data in order to infer something about the content of that text. In some cases, we are looking at a macro level - examining the meaning of the text in a global context (either in terms of phrases, sentences or complete passages). Alternatively we may be interested at a median level (i.e. words). We may also derive valuable data at a lower level, by considering individual phonemes, syllables or characters. The latter method is the one I employed in this thesis. The reasons were purely practical. The aim of this thesis was not to develop new techniques for examining text, but rather to focus on what is done when the analysis determines a failing in the incoming data.

Since we are interested in looking for errors at the earliest possible opportunity (immediately after a key is struck), I chose to evaluate text on a character and statistical basis. Other techniques may ultimately prove more valuable to computer users, particularly those who are disabled, but the importance here is aiding the user to adjust the text when any of the techniques described above finds fault with the text entered thusfar.

Chapter 4 - Digrams and statistics

4.1 Fundamentals

As described in the last chapter, one of the sub-branches of linguistics is the statistical breakdown of text into subsets consisting of pairs, triplets or larger groups of consecutive letters. Some studies have already been performed on the breakdown of English text into digrams (pairs of adjacent letters) and trigrams (triplets of adjacent letters). Tables of the frequency of occurrence in each of these analyses are reproduced in Bailey (1982).

A table of frequency of appearance of individual letters is also reproduced in Bailey, but this is of little practical value in analysing text as we are only interested in the combinations of letters.

It is obvious that digrams are a subset of the trigrams with a space character preceding or succeeding the digram in question and, similarly, that trigrams are a subset of quadrigrams, etc.

4.2 Digrams in particular

Let us begin, then, by examining digrams and their relevance in this thesis. It has already been stated that text can be considered as a long sequence of digrams.

Analysis has already been performed to determine their relative frequency of occurrence in common English text. The value of the latter is of rather dubious nature since it is seldom the case that "standard" text is entered into a computer terminal. Furthermore, an altogether different language may be used. It does, of course, provide some standard on which to base evaluations, but it is clear from the above that a more flexible system of analysing the data entered is required.

When examining text data, several points must be noted. Firstly, only alphabetic characters are analysed. In some cases, numeric characters are used, but we can infer little about their digrammatic content as digits may be regarded as random in terms of likely order of occurrence, so they are excluded from the analysis. Text is punctuated with special characters e.g. the period ".", the comma ",", parentheses "()", etc. The important observation here is that such characters can only (with certain, minor exceptions) signify the end of a word. Thus, they may be regarded as spaces for digrammatic analysis purposes. In general, the difference between upper and lower case bears little consequence, so that case can usually be ignored.

4.3 Statistics

Reference has been made in the preceding section to the relative frequency of occurrence of each digram. This

implies a ratiometric study, i.e. a simple statistical analysis. When analysing the text, the latest digram under examination is being compared with the probability of its appearing at that juncture and deciding whether it is acceptable or not.

At its simplest level, binary probabilities may be used. This means that a digram is accepted only if it is known to EXIST and rejected otherwise. The existence of even the most obscure digrams has to accepted because any uncommon digram would always fail otherwise. An alternative approach is to reject infrequent digrams so that these are always flagged for the user's attention.

It is obvious that this is a less than satisfactory solution. Thus, the next logical level of acceptance/rejection is as follows: If a table is maintained, not merely of whether a digram is known to exist or not, but of its likelihood of being used (a probability level) then a graded decision can be made about whether to accept the digram or not. Most likely, a setpoint for acceptance will be provided and the entered digram compared with its respective setpoint. The major advantage of this technique is that by adjusting the setpoint the contours of acceptance may be altered. If a digram matrix is provided that is carefully designed, dropping the setpoint enables more and more digrams of lower probability to be included to suit the circumstances.

Another advantage is that the statistical contour of setpoints can be altered by updating the levels stored in the matrix so that a completely different set of acceptance criteria is defined. It does not take much of a leap to conclude that we can even set up the software so that it modifies that matrix continuously as text is analysed. Hence, it becomes self- adaptive.

4.4 Practical application

Consider, then, how this information may be utilised in analysing text as it is entered via the keyboard. Without going into any technicalities, I will assume that any characters entered from the keyboard are first passed on to our program which then passes it on to the application they were originally intended to go to. The analysis might then proceed something like this:-

(1) Is this the first character of a word? If not, then go to (2). Otherwise, check the digram <space><character> for validity. This is performed by looking the digram up in the matrix and determining its probability of use. If this statistical level exceeds the minimum setpoint, then the character is passed on. The new character is retained in a temporary store for future analysis. Return to (1).

(2) If this is not the start of a new word, then the first character of a digram is in the temporary store already. If the input is not an alphabetic character, go to (3). Otherwise (i.e. the continuation of a word), check the digram as in (1) above. The new character replaces the one in the temporary store. Return to (1).

(3) If the character signifies the end of a word (i.e. <space> or a special character) then the digram <old character><space> is checked as in (1) above. A <space> is stored in the temporary store. Return to (1).

(4) If the check fails at any point, then the user is alerted in some way. The software would then, typically, offer advice or some alternative choices to the user. At this point, no character would have been passed on to the target program or process, but the selected option (or result of taking the advice) does get passed through. Return to (1).

This process takes place continuously as a background function of which the user is unaware unless the check fails. Further steps may be taken as well as those described above. For example, each final digram (whether it is one that was accepted by the program or one that was over-ridden after advice was offered) may be incorporated

in the store so that the matrix adapts itself as it is used.

Here, now, is the basis of a text analysis program that uses digrammatic tests to verify the validity of entered data. There are many improvements to this basic technique that can be effected, some of which will be considered in the following chapters.

Before concluding this chapter, it may be noted that suggestions have been made that include offering the user "advice" or options. The means I have adopted for the package developed for the thesis is discussed at a later stage, but it is useful to pause and think about the relevance of digrams, their frequency of use and the reasons for the QWERTY keyboard layout at this point. The QWERTY keyboard was designed to reduce the likelihood of adjacent key hammers being struck consecutively (and jamming). This does not refer to adjacent keytops, but there is a certain correspondence between the two which is of great interest to us.

If there were a keyboard layout that performed the function of separating characters which would apply to any previous character sequences equally well, then the likelihood of selecting a valid digram (or probable digram) by mis-striking an adjacent key would be reduced. This is the problem being looked at: the disabled user who frequently hits a key ADJACENT to the target key.

Chapter 5 - Trigrams and further extensions

5.1 Trigrams - after the digram

As described in chapter 4, trigrams are simply a superset of digrams incorporating letter triplets instead of pairs. However, using trigrams extends the text analysis in a number of ways.

Firstly, it is obvious that by considering three letters at a time, we are able to perform a more extensive test of entered data. For example, digrammatic analysis might allow the pair "sk" (ask, dusk, etc.) and the pair "kn" (knife, knight, etc.) and, hence, accept the word **asknce** (instead of askance). Trigrammatic analysis would probably fail this due to the unlikely trigram "skn". Note that this trigram is considered unlikely, not "illegal" despite its non-existence in common English. (e.g. SKN is a command used to load an abbreviated form of SideKick (Borland Software) under MS-DOS.)

Secondly, the use of trigrams spreads the matrix over a far wider base, allowing for more variation in the acceptance profile of different letter combinations. This could allow the user to spread the probabilities for less common trigrams more evenly and prevent the program alerting the user to possible, but unfamiliar, trigrams.

Another effect that is brought out in trigram analysis is that, for words with few enough letters (in the case of trigrams, 1 letter words), the program serves the function of a lexical spelling checker **as well as** standard trigram analysis. As larger groups of letters are used, more words are covered based on the requirement that the polygram can enclose a word with spaces at each end. However, because the program is not limited merely to complete words, the space saving techniques applicable to lexical dictionary compression cannot be used here. See section 5.3 for a more detailed explanation of this compression.

5.2 Incorporating trigrams in the program

The theory of analysis of trigrams follows the same philosophy used for digrams as described in chapter 4. The changes that are required merely take into account that a 3 letter shift register is used for obtaining the trigram. Thus, in step 1, the first letter of a word is checked as <space><space><letter>. Subsequent letters are fed in from the right and when a word is terminated, two trigrams are then checked:

<2nd-last-letter><last-letter><space> and
<last-letter><space><space>.

Since the entered text is always verified against a matrix of pre-collected data, this matrix must obviously expand to

meet a growing number of additional combinations of letter groups (polygrams). It is also important to provide some base matrix from which to operate when the program is first executed, otherwise it will take a considerable time for the matrix to adapt to the style of text usage. A program must be written to analyse some basic text (typical of the intended usage) and generate such a basic matrix. It is evident, too, that suitable code must be produced to access this matrix and update it as it is used.

An inherent problem with the program begins to show itself here. The data file used to store the trigram matrix grows rapidly. For digrams, a 4 kByte file was sufficient, for trigrams, the file is close on 38 kBytes. I will discuss the reasons for this in a little more detail in the next section.

5.3 Quadrigrams, quintagrams...polygrams - where to stop

It is clear from the above sections that there are advantages to be gained from increasing the size of the letter groups used. Lexical checkers could be dispensed with as they would form a subset of the package and would not offer anything like the flexibility. Detailed personal studies on an individual's language use could be performed and "ideal" polygram profiles drawn up. All data entry could be evaluated, checked and assistance offered where required.

Unfortunately, such Utopian software does not conform to reality. Already discussed is the size of a trigram file. A similar quadrigram file would use > 1 MByte (3 floppy disks on a PC) and the mathematical progression is a power one. It is described by the basic formula:

$$N = B \times 27^P$$

where

N = storage space in bytes

B = number of bytes representing the polygram's probability of occurrence

P = width (in characters) of each polygram

Not only does this result in a lot of disk space consumption, but the matrix must be held in memory while the program runs! This is more memory than most PC's have installed. Quintagrams (requiring 28 MBytes) are out of the question.

Lexical checkers have a distinct advantage here. Because of their somewhat limited operation, compression algorithms can be used to fit the dictionary in a smaller space. These compression algorithms usually operate by removing portions of words that are common to a number of them and storing the words in such a way that the removed portion can be easily inferred. They have a second advantage in that they are interested only in the existence of a word,

not its frequency of use. A single boolean variable uses far less space than a number which may have 16 or 32 bits to it.

Another issue is the speed of operation. Since the analysis is done in real time during keyboard entry, it is unobtrusive so long as the program, operated in conjunction with the main (application) program, does not operate more slowly than the typist. The difference in operating time between digrams and trigrams is almost negligible since the differences in the high speed search routines are slight, and the time taken by the rest of the analysis is small. However, if the polygram width is increased, the total time ceases to be negligible.

Unfortunately, solving the two problems mentioned above turns out to be antagonistic in effect. In order to keep data files to a minimum, data compression techniques will have to be applied. The more data provided, the more complex the compression/expansion of data becomes. This increase in complexity involves a large amount of processing time. Consequently, a compromise between these three variables would have to be determined, i.e. between maximum permissible polygram width, minimum data space (in memory and on disk) and best response time.

5.4 Application of linguistics

As discussed in chapter 3, the general application of linguistics is not yet a viable option. However, considering the dilemma presented above, the benefits to be obtained if we reach a complete understanding of natural language may provide a solution. If linguistic rules can be kept to a minimum and the application of these rules does not take too much processing time, then we are able to handle linguistic (i.e. natural language) sub-groups of virtually any size without compromising the response time or using unacceptable amounts of data storage space. By that time, of course, computer technology will have advanced to the stage where response time is unlikely to be compromised in any case.

Chapter 6 - Improving on the keyboard/user interface

6.1 Artificial large keys - key *clumps*

The problem faced by disabled users in particular, is the inability to strike the correct key accurately. For reasons considered in earlier chapters, it was decided, for this project, to make the task of hitting the required "key" easier during editing periods only. Many of the problems related to exact selection of a key can be alleviated using specially designed keyboards (as in chapter 2), but software can often perform a similar task. Many ergonomists have started looking at so- called self-adaptive interfaces (Innocent, 1982).

The problem is that most of these studies have occurred at a high level (Benyon and Maskery, undated). The adaptiveness has been in the general behaviour of the system as a whole, not merely the user interface level. My objective was to modify the user interface only, allowing the user to have the benefit of the general interface of the underlying program with minimal change. The means for achieving this is described below.

When an "error" correction sequence is begun (i.e. the user has accepted the offer of assistance made by the program) then the keyboard is reassigned into large *clumps*. In doing this, I have attempted to make the break-up as

logical as possible, to provide large, even areas and to make available intermediate dead bands where possible.

A *clump* consists of a group of adjacent keys, forming a rectangle or other suitable shape. Typical groups would include:

the function keys	(<F1> to <F10>)
the cursor control pad	(<HOME> through <PG DN>)
the <SPACE> key	(considered large enough to justify its use as a *clump* on its own)
the keys <A>, <S>, <D>, <\>, <Z> & <X>	
the keys <9>, <0>, <->, <O>, <P> & <[>	

and so on. Dead bands include the remote keys such as <SCROLL LOCK>, <RETURN>, etc. as well as the keys <4>, <R>, <F>, <C>, <7>, <U>, <J>, <N>, etc. This principle is graphically illustrated in Fig 6.1.

Thus, the keyboard is divided into six main *clumps* (in the QWERTY section of the keyboard) and three utility *clumps* comprising the other sections of most IBM keyboards.

A problem that arose with testing subjects with TEACH was the variation that exists in keyboards. Allowing for application to only QWERTY keyboards (although TEACH has provision for remapping keyboards - e.g. to DVORAK layout)

there exist at least 4 variations of keyboards on which TEACH has already been used: the original IBM PC layout, the AT extended keyboard and the new enhanced layout, as well as the Olivetti M24 extended layout. The variations are largely limited to positioning the external key groups that perform cursor movement, terminal control and the function keys. Minor variations in the QWERTY section also resulted in a few problems. The layout variations required separate, installable keyboard drivers that provide a different graphic layout on screen as well as alterations in the breaking up of *key clumps*.

6.2 What the *clumps* are used for

The six main *clumps* are used to represent alternative choices. If an inconsistency is detected, the six adjacent keys to the one actually struck are taken into consideration. Those that are valid options (i.e. would not have generated an "error" condition) are offered as alternatives which the user may select to REPLACE the last key struck.

Utility *clumps* offer the extra options:

- original key accepted (as if assistance was refused) and
- other key intended (a key that has not been offered as an alternative).

The user selects an option by striking one or more keys lying in the relevant *key clump*.

6.2.1 An example of this effect in operation

In order to clarify the principle described above, an example is presented here. Refer to this example in future sections as the operation is described in detail.

While editing a document, the typist has entered the following text :-

Professor Johnston fingw

The trigram "ngw" is most uncommon in the English language. The program gives a warning beep and offers the message

"ngw" not accepted. Hit space for help, any other key to ignore.

The typist hits <SPACE> and another window appears offering the following options :-

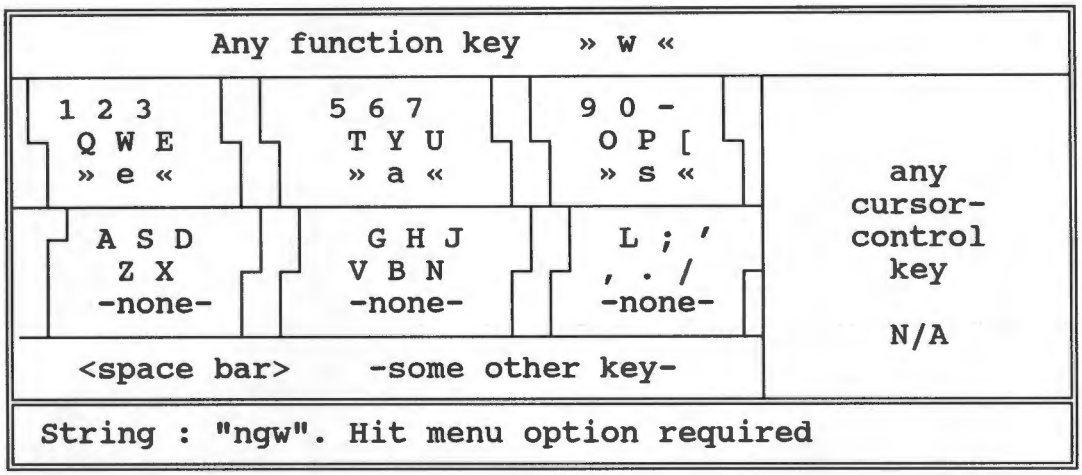


Figure 6.1 - T.E.A.C.H window

The typist now hits any one or more of the keys constituting the *clump* 1, 2, 3, q, w, e in order to get the intended "e" and the window vanishes, leaving the screen as before, but with the text now being

Professor Johnston finge

ready to be completed as

Professor Johnston fingered his moustache ponderously.

6.3 The software version

Implementing these *clumps* in software is relatively easy. Once the user has accepted assistance, a window is

displayed in the centre of the screen to show the relevance of each *clump* (as in Figure 6.1).

The program then gains total control of the keyboard and waits for input. The program then uses a simple menu approach to determine the effect of the first keystroke encountered and other keys are ignored.

At this stage, the program has the original ("erroneous") keystroke stored and now has the option stored as well. If the option selects a different letter (i.e. a valid, adjacent key) then this replaces the original keystroke in the same case as the original. If the original is re-accepted, it is left unchanged. If a different key was required altogether (not a proffered choice), then a NUL keystroke is returned, effectively negating the effect of entering a keystroke. The mechanism by which this operates is described in the next chapter.

The window is now removed and control returned to the foreground program with the chosen character passed on to it. The program returns to its role of watchdog, looking for the next inconsistency.

6.4 Informing the user of inconsistencies

When an improbable polygram is detected, the user must be alerted and given the option of accepting or rejecting further assistance. At this stage, the foreground (main) program is still apparently in full control. In fact, the analysis program has gained temporary control and is ready to hand it back if no assistance is requested.

The warning to the user must be as unobtrusive as possible. To date, the choice has been to pop up a small window indicating that an error has been detected and instructing the user to press <SPACE> for further help, any other key to continue. A short duration beep is also emitted from the speaker in case the user is not watching the screen.

If any key other than <SPACE> is entered, the window vanishes and the character last selected (before the interruption occurred) is passed on as if the analysis program was never invoked. It would be preferable if the key entered in answer to the small alert message was used to indicate: "accept the last key and use this one as the next letter I type". This would enable the user to ignore the program completely if so desired without having to press "any other key to continue". However, this presents software difficulties which have not been solved as yet.

If <SPACE> is entered in response to the alert prompt, then the little window vanishes and a big window pops up in the centre of the screen. This window shows the new keyboard layout using the *clumps* and prompts the user to select a choice. If any dead band key is depressed, the program responds with a warning beep from the speaker and waits for another attempt. Otherwise, the action follows the description in section 6.3 above.

6.5 Additional software functions

Since the *clumps* comprise six or more keys (excluding the <SPACE> key *clump*), there must be provision made for more than one key being struck. If additional keys were ignored by the program, then the keyboard buffer of the PC would store them and pass them on to the foreground program. This is clearly undesirable and some debounce facility must rather take place.

With the *clump* containing several keys, successive strokes on the part of these several keys is effectively a bounce effect, although not technically so. This multiple strike effect has to be eliminated. The simplest way of performing this function is to wait a short period (a fraction of a second should be enough) and then clear the keyboard buffer before handing control back to the foreground. There are alternative techniques, but they are more complex and, generally, more demanding in their

criteria. The problem with this particular approach is to determine the duration of the delay before further keystrokes are considered intentional.

It has already been stated that only valid alternatives are offered in the main window. These alternatives are those adjacent keys to the one struck which would have passed the check. Other approaches may be considered valid. Perhaps ALL adjacent keys should be offered, irrespective of whether they are valid or not. Alternatively, perhaps only the most likely polygram terminators (letters considered most valid) should be offered without having to be adjacent to the original keystroke. Perhaps the unused locations in the current approach (left by adjacent keys which are not valid) should be filled by next-to-adjacent keys which are valid, etc.

There are clearly many approaches. One could insist that all locations be filled by keys, whether TEACH considers them appropriate or not. Alternatively, TEACH could fetch appropriate options from further afield than the adjacent keys. More examination of such alternatives is considered in Chapter 8. Just one of them will suffice for the purpose of this thesis. Others could be considered and implemented later.

Chapter 7 - Testing the effects

7.1 The methods used

In order to evaluate the effects of the program on a user, I assembled some passages of text previously used for this purpose and had some subjects enter these passages using a specialised "word processor", modified for the evaluation. The passages are listed in Appendix D.

The editor used was a modified version of the BINED text editor that forms a part of the Borland Turbo Pascal Editor Toolbox. The modifications effected were means to time the duration of the editing session, to count the keystrokes entered and to return the resulting typing rate. The editor also provided a means of loading the test program (TEACH) for the duration of the edit session and to report how many times TEACH offered assistance and how often it was accepted.

I also counted the number of mistakes that were found in each passage after it was typed and attempted to differentiate between typographical errors and those that may be culturally or linguistically based (for example, some people erroneously spell "friend" as "freind").

The results are not fully representative of those that might be achieved with any passage and database

combination. I compiled a database based on the passages in Appendix D only. The result was a database that would provide at least 2 warnings while the word "HELLO" was entered. It allowed a complete, accurate entry of any of the supplied passages without providing a single warning. It would also flag most deviations from the passage (except where words are left out, words are added that exist elsewhere in the text, or words are modified in a way that matches other word usage - e.g. doorway -> doorwalk).

7.2 Results - candidate 1

The results of performing this test on a person with C.P. are given below. The candidate in question presented with the same difficulties as the person with C.P. around whom the program was originally written, although the incidence of mistakes was far lower with this candidate. The reasons for this lie in the severity of the disability. This particular person had been typing for some time (and had completed a typing course) and had developed some proficiency through the use of a keyboard mask (a plate mounted above the keyboard with a hole at the position of each key).

This candidate typed in the passages numbered 2 (without TEACH) and 3 (with TEACH) after familiarising himself with the editor and TEACH on the other two passages.

7.2.1 Trial 1 - Passage 2

Total number of letters in the passage :	660
Total keystrokes needed to enter passage :	808
Total alphabetic keystrokes in file :	664
Total characters in resulting file :	819
Time to type passage (seconds) :	1265.04
Typing rate (characters per second) :	0.62
Total errors recorded :	14
Typographical errors recorded :	10

7.2.2 Trial 2 - Passage 3

Total number of letters in the passage :	659
Total keystrokes needed to enter passage :	790
Total alphabetic keystrokes in file :	654
Total characters in resulting file :	789
Time to type passage (seconds) :	1396.76
Typing rate (characters per second) :	0.56
Calls to TEACH :	703
Times help was accepted from TEACH :	21
Total errors recorded :	4
Typographical errors recorded :	2

7.3 Results - candidate 2

Below are the results of performing the trials on a partial quadriplegic subject. The person in question has

a T6 lesion, resulting in limited use of his arms, wrists and fingers. He types using the back of the second joint on his right hand small finger only. It is necessary for him to place the keyboard in his lap and suspend his right arm above the keyboard while positioning his finger over the desired key. He then drops his hand onto the keyboard, striking the key in the process, before retracting his entire arm. His typing is remarkably accurate, although very slow and tiring.

The candidate typed in passage 3 without TEACH assistance and passage 4 with TEACH.

7.3.1 Trial 1 - Passage 3

Total number of letters in the passage :	659
Total keystrokes needed to enter passage :	790
Total alphabetic keystrokes in file :	656
Total characters in resulting file :	782
Time to type passage (seconds) :approx.	1380
Typing rate (characters per second) :	0.57
Total errors recorded :	10
Typographical errors recorded :	8

7.3.2 Trial 2 - Passage 4

Total number of letters in the passage :	713
Total keystrokes needed to enter passage :	887
Total alphabetic keystrokes in file :	722
Total characters in resulting file :	888
Time to type passage (seconds) :approx.	1845
Typing rate (characters per second) :	0.48
Calls to TEACH :	760
Times help was accepted from TEACH :	12
Total errors recorded :	10
Typographical errors recorded :	3

7.4 Examination of results

Analysis of this data in terms of practical consequences is difficult as a result of the limited number of subjects tested. The subjects whose results appear above were the only subjects who could be found that provided results that could be measured, as other subjects were unable to perform the tests because of limited cognitive capacity or memory function.

I have decided that the two most important factors that should be measured are typing rate and error count. The intention of TEACH is to provide assistance in correcting errors in two ways :-

(a) alerting the user to typographical errors reduces the final number of errors and speeds up the correction of some other errors by allowing the user to edit the data as soon as possible (resulting in a minimum amount of file manipulation to get to the error);

(b) speed up the editing process in many cases by making it much easier to select correction options.

The tests have conflicting results. The typing rate is slightly reduced when using TEACH. This may be partly attributable to the need to read a popped up window of data and to search for the appropriate option in the "menu" offered, and also to unfamiliarity with using TEACH. While watching the users enter the data, I observed that they would pause briefly when TEACH issued the warning beep before examining the text that had just been entered and flagged as unlikely. This was particularly visible in the case of candidate 2. This subject also experienced a different problem. He would frequently hear the beep warning him of a detected mistake, but it would be too late to prevent him from typing the next key. The result was that he would then miss out on getting help from TEACH and have to correct the error himself, with no assistance. The wasted time was then compounded by him trying to get TEACH to help by

pressing the space bar - entering an undesired space character or two.

On the other hand, the typographical error count is substantially down in both cases. This is consistent with the fact that TEACH will warn the user of most typographical errors in the passages used for this evaluation.

Many would argue that the error improvement far outweighs the slight loss of typing speed. It may also be argued that the rate will improve with familiarity with the use of TEACH. Improvements in the user interface could reduce the time spent searching the options for the desired operation and in providing a wider range of useful options.

On the other hand again, it must be remembered that the TEACH database was the ideal one for these passages. It would not provide any false warnings and would catch most errors. In normal use, TEACH could miss far more errors and give a number of false warnings.

7.5 Other potential subjects

It should also be noted that at least 20 other subjects tried using TEACH. Their results have not been reflected here since, in the few cases where they could be measured,

they were not generally meaningful. In some cases, the users proved unable to grasp the concept of TEACH and in many more, the concept of editing and using a computer proved to be too demanding! There is a common perception that computers are for highly intelligent people only and this is particularly prevalent amongst users with disabilities who may also be intellectually disadvantaged. The perception leads to these subjects convincing themselves that they will never cope. They often demonstrate themselves to be incapable of using a computer in order to avoid challenging their own *status quo*.

In particular, a number of subjects who have Multiple Sclerosis (M.S.) were tried. With 2 or 3 exceptions, they proved cognitively incapable, suffered such serious memory lapses that they could not remember how to use the computer for long enough, had emotional instabilities that interfered with their use of the computer or were otherwise reluctant to even try. (Note that I am unable to confirm which of these difficulties were a result of M.S. and which were unrelated.) Those that did try experienced no positional accuracy problems, not being afflicted with the "intention tremor" that often accompanies this illness.

Many of them were unable to come to grips with a "Typematic" keyboard. (A "Typematic keyboard is one that begins to auto-repeat a key after a while if it is kept

depressed.) Since the trials were performed on a lap-top computer, it was not possible to overcome this problem.

7.6 Unsuitable subjects

It becomes clear from the attempts to test the applicability of TEACH to people with M.S. that TEACH has a very limited applicability. The number of users with disabilities who could benefit from the assistance that TEACH offers is a very small proportion of the overall disabled population. In fact, the users who would benefit include only a small range of those with motor co-ordination difficulties. Severely disabled people would be unable to use a conventional keyboard and those with minor disabilities might find that TEACH simply slows them down. If the co-ordination problem is compounded with cognitive disability, the interface presented by TEACH becomes too difficult to comprehend.

In addition, it was thought in the early stages of TEACH's development that the program might find use with non-disabled computer users. However, use of the program myself (while testing) indicated that TEACH would actually be severely disadvantageous to experienced programmers and computer users as it slows down keyboard usage considerably. I would guess that it would also take a very short time for novices to begin experiencing similar disadvantages. Early experiments with novice, non-

disabled users showed that TEACH was more likely to engender bad typing habits. This came from knowing that the computer would probably catch errors, so there was no incentive to type accurately.

Chapter 8 - Conclusions

8.1 Evaluation of TEACH as it stands

The results presented in the last chapter appear to indicate that TEACH does not provide a major improvement, if any, to the typing of a particular group of disabled users. Clear reasons for its failure to provide good results in its present form are discussed below. Providing the improvements that I shall suggest may or may not produce much better results, but there are important observations to be made from the evaluation in any case.

One major point is that keyboards as they currently stand are inadequate for a many disabled users. Keys are difficult to strike accurately and repetitively when they are closely spaced and small. Many non-disabled users have difficulties using keyboards (on many occasions I hit more than one key at a time myself!). This problem is drastically increased in the case of people with positional accuracy and co-ordination problems.

TEACH also provides an alternative to forcing the disabled user to purchase custom, but expensive, keyboards. In all cases where users with disabilities tried to use the TEACH package, they expressed approval of the techniques used. For some, it was merely a case of being dazzled by the windowing and software techniques, but in the main,

subjects approved of being able to treat the keyboard of a conventional computer as a collection of "large keys" or key clumps.

8.2 Possible areas of improvement in TEACH

Currently, TEACH will alert the user to a bad key choice (resulting in an unlikely trigram) even if it cannot provide any corrective options (because any keys that would make sense are not adjacent to the one last selected). For example, pressing <Q><Q> would cause TEACH to query the trigram, but no valid keys are in range to be offered as alternatives. TEACH should certainly flag the error under these circumstances, but should either suppress presentation of the menu, or present a range of options by drawing possible combinations from more distant keys.

TEACH presently provides empty menu entries where options are not available. Again, further ranging, semi-adjacent keys should rather be offered here. TEACH also fills the menu spaces on a first come, first served basis, depending on the order of data in a supplied keyboard definition file. It would be beneficial to arrange the keys in the menu in a way that is representative of their physical placement on the keyboard. This would limit the amount of searching required to find the correct option in the window. To illustrate this discussion, examine Figure

8.1, below. It displays a section of the qwerty keyboard with the keys surrounding the <H> key. Let us say that the <H> was most recently struck and TEACH disapproved. It considered the letters <G>, <N> and <Y> appropriate. Currently it would offer the options shown in Figure 8.2. However, we might consider the window in Figure 8.3 more appropriate.

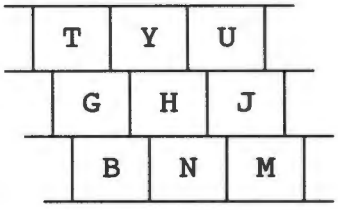


Figure 8.1 - QWERTY keyboard extract

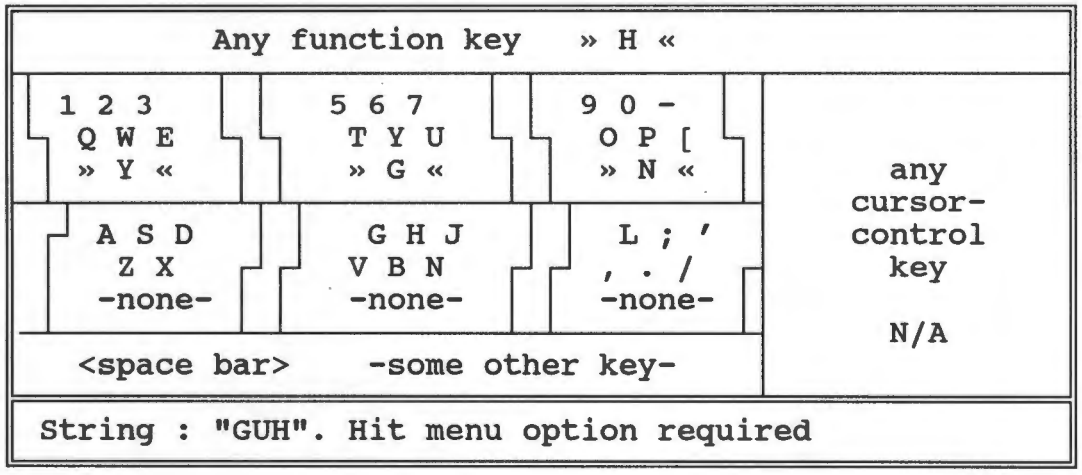


Figure 8.2 - Standard T.E.A.C.H. options

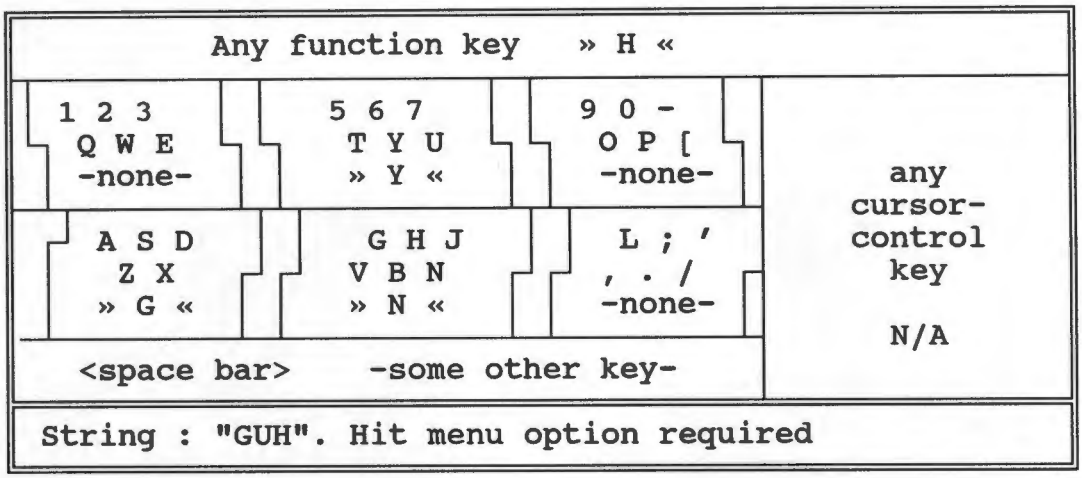


Figure 8.3 - Improved T.E.A.C.H. options

It is only possible to get TEACH assistance if TEACH has detected an error. Many users expressed the desire to get at the TEACH window when they detected an error that TEACH had not picked up. This would allow an improvement in correction time. It may also be advantageous to have TEACH offer suggested "next" letters. For example, if the user has entered "hel" so far, calling TEACH may allow the next "l" of "hello", the "p" of "help" or other appropriate characters. This could allow for a general improvement in typing speed.

TEACH uses a very simplistic technique for analysing text as it is entered. Borland International has a package called Word Wizard that allows Pascal programmers to access Turbo Lightning's dictionary. It may be practical to use the supplied dictionary or a specialised dictionary that provides checking of partial words to provide a better analysis technique. This would also allow TEACH to

offer replacement words where errors crept in at the beginning of a word and not as the last letter only. A different, far superior method would be the application of research into natural language processing when results become available.

Since I embarked on writing TEACH, I have learned a great deal about the coding of Terminate but Stay Resident (TSR) programs (see Appendix B) and the interrupt services available on the IBM PC and compatibles. What this means, is that I have since learned better ways of allowing a TSR to interface to a foreground task. Thus, it would be possible to provide multiple letter options as a response to TEACH detecting and correcting an error. At present, TEACH can only replace the latest letter. If the user chooses to abandon the character, TEACH returns a NUL key to the main task. Some programs treat this instance in a special way - in many cases considering it a special command! This problem could now be avoided by altering the way in which keystrokes are fed to the application program.

TEACH also presents another difficulty, often related to the Typematic keyboards. When it initially warns the user of a dubious text entry, it waits for a key to be pressed. Sometimes the Typematic keyboard fools it by repeating the key that was erroneously entered. Since this is a letter, TEACH retracts its offer of assistance and the opportunity

is lost. In other cases, the user types two letters rapidly in succession, hearing the beep too late to catch the last entry. Again this means the loss of TEACH assistance. Solutions for this are not immediately obvious. Perhaps it is necessary to appoint specific keys or clumps for the user to decide whether assistance is required or not. TEACH would then insist on one of these keys before continuing.

8.3 Other lessons learned from TEACH

A problem found with present keyboards that hasn't really been discussed previously is that of "Typematic" operation. Many users found that they kept on producing multiple keystrokes because they could not lift their fingers from the keyboard quickly enough! This problem can be reduced on more recent computers (the AT class of machines) by reducing the repeat rate and increasing the time before key-repeat operation begins, but the problem cannot be eliminated entirely. It may be possible to eliminate this problem with software, but the means is not immediately obvious. While this problem is not a problem with TEACH itself, it is nevertheless an area that needs attention lest it prevent programs like TEACH being used. This could mean correcting the deficiency in these computers within TEACH, or it may mean a separate program.

Ideally, it should be possible to change any of these characteristics of a computer within the hardware. The programs used to drive such units as the keyboard controller in a PC are not alterable (this is an immutable and limiting factor at this stage) and are not written such that the characteristics of the device's operation can be changed (the "firmware" in the device could take this into account to some extent).

It would also be possible to eliminate such problems if the Basic In/out operating System (BIOS) of the machine was adaptable. (The BIOS refers to a program that is encoded into some of the hardware of the computer.) Unfortunately, the current specifications for a PC compatible's BIOS does not force the hardware designer to conform with the original hardware design, so that it is not practical to rewrite selected BIOS functions to suit the current need (such adapted software could not be guaranteed to work with all machines). Furthermore, the BIOS was written without consideration for these needs. Personal computers were designed for non-disabled users. People with disabilities were not taken into account.

8.4 Final conclusions

Even if TEACH should never prove to be of direct benefit to users with disabilities in itself, I hope that it highlights a number of the problems that disabled users of

computers must face because of inconsiderate design of hardware and software in the PC marketplace today. I hope, too, that it encourages the producers of this equipment and the applications that run on it to examine the usability of their products by **all** members of the population. Many disabled people are in a position to exploit their otherwise hidden abilities thanks to the advent of the personal computer era. Sadly, most of them are locked out of this wonderful opportunity because they are unable to interface to even the simplest of applications.

A simple, but important suggestion is that future personal computer designers (and designers of related equipment) include the ability to have hardware deficiencies in their systems over-ridden by add-on software. In this area, the Apple Macintosh leads the way. I hope that others will soon follow.

Indeed, the keyword here is *INTERFACE*. In many centres, this subject is dealt with under the heading of *Ergonomics*. Unfortunately, these same centres that advocate the need to interface modern technology to users transparently, neglect a large group of potential clients. The position of the person with disabilities is invidious enough without being excluded from potential benefits of the scientific age.

It is also a fervent hope of mine that the work I have performed will encourage others to attempt to alleviate the problems of people with disabilities by producing products that allow them to use the popular applications already available, until such time as the applications address the problem themselves.

Appendix A - List of References

Angermeyer J, Jaeger K, et al.
1989
The Waite Group's MS-DOS Developer's Guide.
Howard W.Sams & Company, Indianapolis

Bailey RW.
1982
Human Performance Engineering: A Guide for System Designers.
Prentice hall, Englewood Cliffs NJ

Baran N.
1988
An Analyst for Your Writing.
Byte Magazine, August 1988:92-94

Benyon D.
(date unknown)
MONITOR. A self-adaptive user interface.
Unpublished paper - University of Warwick

Butler C.
1985
Computers in Linguistics.
Basil Blackwell Ltd., Oxford, Chapter 2

Crystal D.
1985
Linguistics.
Pelican Books, Middlesex

Co-ordination committee: Year of Disabled Persons 1986
1987
Disability in the Republic of South Africa
Department of National Health and Population Development, Vol 1

Gilad I & Pollatschek MA.
1986
Layout simulation for keyboards.
Behaviour and Information Technology, 5(3):273-281

Innocent PR.
1982
Towards self-adaptive interface systems.
Int. Journal of Man-Machine Studies, 16:287-299

Johnson EL.
1986
Keyboards for the Handicapped.
Journal of Medical Systems, 1(3):277-287

Krammerer E.
1988
SOUNDEX.PAS & SOUNDEX.TXT
Turbo User Group Disk #MI-CCC-DOS-033-I5 : demonstration program and text

Lemmons P.
1982
A Short History of the Keyboard.
BYTE Magazine, 7 November:386-387

McDougall J, Knysh B, at al.
1988
Computer-based Technology for Individuals with Physical Disabilities:
Guidelines for Alternate Access System Developers
University of Toronto

Macaulay LA & Norman MA.
Designing interfaces for different types of users - Experimental criteria.
Internal design papers

Maguire M.
1982
Computer recognition of textual keyboard inputs from naive users.
Behaviour and Information Technology, 1(1):93-111

Marcus MP.
1982
A Theory of Syntactic Recognition for Natural Language
Artificial Intelligence: An MIT Perspective
The MIT Press, Cambridge MA

Maskery HS.
(date unknown)
Adaptive interfaces for naive users - An experimental study.
Unpublished paper

Pollatschek MA & Gilad I.
1986
The customizable keyboard.
Behaviour and Information Technology, 5(3):283-287

Waller A.
1989
Implementing Linguistic Text Anticipation in a Writing Device for the Disabled.
Dissertation for MSc(Med) Biomedical Sciences. University of Cape Town.

Appendix B - Resident code

B.1 Background tasks

The advent of the modular programming approach used on IBM PCs and compatibles (amongst others) made possible the concept of background task operation. While the technique is not new, it is generally true to say that it was only implemented previously on older, single-user machines when the software was a complete package written to operate in such a mode. Only recently has it become a simple task for user written software to be able to employ this feature without interfering with the main program executing on the computer at the same time.

The technique allows a form of multi-tasking to take place. The mode of task scheduling is dependent on how the program is implemented but is generally one of the following:

- True time division multi-tasking which uses a regular time based signal to change the task currently executing.
- Event initiated task swapping which requires some particular stimulus to begin operation of a particular task.

- Hierarchical task sequencing which performs most important tasks immediately and progresses down the priority list.

In the particular case of the software written for this thesis, the second method is used. The stimulus is the entry of a character from the keyboard. When this occurs, the background task is initiated and allowed to run until it has completed its operation (analysis of the text entered thusfar and the user specified response <if needed> to the program prompts <if any>). While it is operating, it appears to have complete control of the machine. However, other multi-tasking operations may also occur using other sequencing methods or other stimuli to initiate the task, or this task may be one in a chain of keyboard interception routines that swap operation on a number of occasions.

An example of the former is the standard MS-DOS PRINT command. This command loads some resident code and uses a timer based interrupt to take control briefly every eighteenth of a second, or so, and send a few characters to the printer, the number of characters being determined by the relative availability of CPU time. Examples of the latter include Borland's utility packages SideKick and SuperKEY and similar programs. They sit in a chain that passes the last keystroke from one to the next until it reaches the task that intends to use that keystroke or it

passes out at the end to be used by the main program executing.

To the user, these tasks all seem to operate simultaneously. While entering data, awareness of only the foreground task may exist until an inconsistency is detected. Then it becomes apparent that the background task has been active all the time, monitoring input and reacting to it in an appropriate fashion. When it takes control, however, it suspends the foreground task until it has completed its own operation.

B.2 Shortcomings of multi-tasking

Many problems can occur when multi-tasking is incorporated. The most obvious is that the foreground does not have complete control of the machine (the background must get it sometime!). Hence, a bug in the background task could destroy all data recorded by the foreground and many hours of work. If the communications between the two tasks fails or the hand-over is not clean, the same could occur.

The time required to run both tasks serially (as they must - there is only one CPU) may exceed that which is available. Time out errors may result and one or both tasks may be aborted with potentially disastrous results.

The multi-tasking mode of operation relies on the operating system (e.g. MS-DOS) to arbitrate and manage the sequencing and if this is not handled correctly, problems will very likely occur. This is a particularly serious problem since, in many cases, complete documentation is not available and much work is carried out in the dark or with word-of-mouth knowledge. Further problems result when using a so-called compatible because such machines often have slightly different operating systems that produce different results and choosing a middle of the road approach that will be machine independent is difficult. This complication applies to more than just multi-tasking, but the latter is a more stringent requirement and will probably be the first area to present problems. MS-DOS is also non-reentrant in many cases (i.e. if MS-DOS is interrupted it should not be requested to perform any functions during the break without special precautions being taken) (Angermeyer, Jaeger, et al., 1989).

Other tasks may have to execute and they could introduce any or all of the above complications. Since they are probably operating on a different type of scheduling approach, they may have other, peculiar effects that are difficult to determine or predict.

B.3 Software adaption to various environments

Once the software is written to operate as a background task, a level of control is lost to the program itself. It is now required to co- operate with other software packages. This presents a problem in itself. Namely, if different packages use different editing keystrokes, how does the program ensure that it sends the correct characters to the particular foreground operating? This question is particularly relevant if linguistics are going to be incorporated at some stage since it may be necessary to remove a portion of the preceding word and replace it with something else (e.g. conversion of "their" to "there", etc.).

Thus, some form of "environment awareness" must be included. It must be possible for the user to specify a foreground task name and the program must configure its communication routines to suit the environment.

Since it is not a necessary feature for the operation of the package as it stands, this feature has not been incorporated into the code yet, but the concept must be acknowledged if the program is to be extended in the future.

B.4 Residency

In the MS-DOS machines, true multi-tasking is not an intended feature of the operating system, but the approaches described above are underhand methods of forcing the system to operate in such a mode. Since it does not have true task scheduling and hand-over management facilities, some means has to be found for the software to load itself and remain loaded so that execution can be passed to it with a minimum of fuss, and for the software to perform the necessary task management.

The former is a capability that has been included in more recent versions of MS-DOS (version 2 and up). The "Terminate but stay resident" function allows a program to load, stop executing and inform MS-DOS just before doing so that this is its intention and that space should be reserved for it to stay in memory and have a variables storage area. This is termed "residency" and such programs are referred to as TSRs (Terminate but Stay Resident). Care must be taken when loading such programs that they do not conflict with other programs for memory or pseudo-multi-tasking entry handles[2] (interrupts in

---

[2] The term "handle" is a computer programming definition that refers to a software vector left in memory where standard programs know to look, in order to find an address to hand execution over to (hence, HANDles). In MS-DOS, these are usually found in the form of software interrupt vectors. More details on the use of these techniques are usually to be found in texts on operating systems.

MS-DOS). The memory management is a function of MS-DOS itself, but the user must be warned that loading resident programs which are **not** required should be avoided to allow as much free space as possible for those which **are** used.

As far as reserving the required handles is concerned, the package written for this thesis (TEACH - Text Entry Analysis and Correction Help) will check to see if the handles it wishes to use are available or if they have already been allocated. If they are free, it will load with no problems, otherwise it will provide a message stating that it cannot load and will abort. **Other resident programs may not be so considerate!** Some other routines may just go haywire and cause the machine to hang, while others may just grab the handles, even if they are in use. This will prevent the original resident code from ever getting the chance to execute and may produce unpredictable results. Caution is always advised when dealing with resident programs.

Appendix C - Listing of TEACH program

```
Program TEACH;

{$M 4000,16000,50000}

uses Crt, DOS;

{ * * * * * * CONSTANTS * * * * * * * * * * * * * * * * * * * * * * * * * * * * * }

const
    Our_Char : byte =  113;   { this is the scan code for AltF10   }

    Ctrl_Home       = #119;   { Control Home Scan Code             }
    Quit_Key        = #119;
    Ctrl_End        = #117;   { Control End Scan Code              }
    Kbd_Int         = $16;    { BIOS keyboard interrupt            }
    Vseg            = $B800;  { CGA video segment - change for MDA }
    HelpColor       = $1B;
    Alphabet : set of char = ['A'..'Z','a'..'z'];


{  - - - - - - T Y P E   D E C L A R A T I O N S - - - - - - - - - - - - -  }
type
  FilenameType  = string[64];
  TrigramMatrix = array[1..27,1..26,1..27] of integer;
  KbdType       = array[1..26,1..6] of integer;
  RoundKey      = array[1..6] of char;
  WindowPointer = ^WindowVar;
  ByteMatrix    = array[0..127] of byte;
  WindowVar     = record
                    WindowXpos   : integer;
                    WindowYpos   : integer;
                    WindowWidth  : integer;
                    WindowHeight : integer;
                    WindowMatrix : array[1..25,1..160] of byte;
                  end;
  ContentArray = array[1..25] of String[80];
  InsertArray  = array[1..8] of record
                                  x, y : byte;
                                end;

{ - - - - - G L O B A L   V A R I A B L E S - - - - - - - - - - - - - - - - - }
var
  Terminate_flag, Done          : boolean;
  OldX, OldY                    : integer;
  Keychr, Keystroke             : char;
  Dummy, Previous, MorePrevious : char;
  Matrix                        : TrigramMatrix;
  Keyboard                      : KbdType;
  Prompt,HelpChars              : ContentArray;
  HelpSize                      : record
                                    X, Y : integer;
                                  end;
  PromptWindow                  : WindowPointer;
  ScanCode                      : ByteMatrix;
  OldInt16                      : pointer;
  KeyScan, Count                : byte;
  KeyInsert                     : InsertArray;
```

```
procedure Get_Abs_Cursor (var x,y :integer);
{-------------------------------------------------------------------------}
{ Get_Abs_Cursor returns the current cursor co-ordinates (faster than Turbo)  }
{-------------------------------------------------------------------------}

var
  Active_Page  : byte absolute $0040:$0062;  { Current Video Page Index}
  Crt_Pages    : array[0..7] of integer absolute $0040:$0050 ;

begin
  X := Crt_Pages[Active_page];      { Get Cursor Position       }
  Y := Hi(X)+1;                     { Y gets Row               }
  X := Lo(X)+1;                     { X gets Col position      }
end;


procedure Blip;
{-------------------------------------------------------------------------}
{  Blip emits a short note on the speaker for audio indication of a detected  }
{  probable error that may be "corrected" or ignored.                       }
{-------------------------------------------------------------------------}

begin
  sound(200);
  delay(20);
  nosound;
end;


procedure Stay_Xit;
{-------------------------------------------------------------------------}
{  Stay_Xit Removes the resident portion of the code and restores DOS.      }
{                                                                           }
{  Clean up the Program ,Free the Environment block, the program segment    }
{  memory and return to DOS.                                                }
{-------------------------------------------------------------------------}

var
  Regs : registers;

begin
  Blip;
  delay(100);
  Blip;

  SetIntVec($16,OldInt16);      { Restore old interrupt vector      }

  Regs.AH := $49;               { Free Allocated Block function     }
  Regs.ES := MemW[PrefixSeg:$2C];   { Free environment block        }
  MsDOS(Regs);

  Regs.AH := $49;               { Free Allocated Block function     }
  Regs.ES := PrefixSeg;         { Free Program (i.e. PSP)           }
  MsDOS(Regs);

end  { StayXit };
```

```
{$I WINDOWS.INC}
{-------------------------------------------------------------------------}
{ The WINDOWS include file contains window management routines to allow    }
{ dynamic opening and closing of windows (using the heap) and moving of    }
{ windows around the screen.                                               }
{-------------------------------------------------------------------------}


procedure Readkeyfile(var keys : KbdType; var codes : ByteMatrix);
{-------------------------------------------------------------------------}
{ Readkeyfile reads the keyboard parameters from external files            }
{                                                                          }
{ The user is requested to identify the keyboard being used (type, layout,..) }
{ and the localkeys Matrix is read as well as the scan code Matrix (required  }
{ for resident operation since these factors have to be passed on to the next }
{ resident keyboard handler in the chain)                                  }
{-------------------------------------------------------------------------}

var
  FileName    : string[30];
  KbdFile     : file of KbdType;
  CodFile     : file of ByteMatrix;
  OK          : boolean;
  OldX, OldY : integer;

begin
  FileName := '';
  write('Enter keyboard filename (no extension) [Blank = QWERTY] : ');
  Get_Abs_Cursor(OldX, OldY);
  repeat
    gotoxy(OldX,OldY);
    clreol;
    readln(FileName);
    if FileName = '' then FileName := 'QWERTY';
    assign(KbdFile,FileName);
    {$I-} reset(KbdFile) {$I+};
    OK := IOResult = 0;
    clreol;
    if not OK then writeln('Cannot find file "',FileName,'". Try again.');
  until OK;
  read(KbdFile,keys);
  close(KbdFile);
  FileName := FileName + '.COD';
  assign(CodFile,FileName);
  {$I-} reset(CodFile) {$I+};
  OK := IOResult = 0;
  if not OK then
    begin
      writeln;
      writeln('Cannot find file "',FileName,'"');
      writeln('Program terminating...');
      halt(1);
    end;
  read(CodFile,codes);
  close(CodFile);
end;
```

```
procedure ReadTrigrams;
{-------------------------------------------------------------------------}
{ ReadTrigrams gets the trigram Matrix from an external file.             }
{                                                                         }
{ The user is prompted for a trigram file name (typically TRIGRAM.xxx) and }
{ is read into the Matrix. Different trigram files may be used to suit the }
{ user and the environment. This file is derived using MAKETGMS.COM.      }
{-------------------------------------------------------------------------}

var
  TrigramFile : file of TrigramMatrix;
  OK          : boolean;
  TriFile     : string[30];
  OldX, OldY  : integer;

begin
  write('Enter the trigram file name (blank = TRIGRAMS.1) : ');
  Get_Abs_Cursor(OldX,OldY);
  repeat
    gotoxy(OldX,OldY);
    clreol;
    readln(TriFile);
    if TriFile = '' then TriFile := 'TRIGRAMS.1';
    assign(TrigramFile,TriFile);
    {$I-} reset(TrigramFile) {$I+};
    OK := ioresult = 0;
    clreol;
    if not OK then writeln('Cannot read file "',TriFile,'". Try again.');
  until OK;
  read(TrigramFile,Matrix);
  close(TrigramFile);
end;

function Screen(XCoord,YCoord : integer) : char;
{-------------------------------------------------------------------------}
{ Screen returns the character on the screen at the specified coordinates. }
{-------------------------------------------------------------------------}

begin
  Screen:=chr(Mem[Vseg:(XCoord shl 1) - 2 + (YCoord-1) * 160]);
end;
```

```
procedure Restore_Previous;
{-------------------------------------------------------------------------}
{ Restore_Previous sets up the past history variables.                    }
{                                                                         }
{ The screen is examined to determine the two characters prior to the latest  }
{ one being serviced so that validation of the trigram may be performed.      }
{ Previous and MorePrevious are restored if they exist and are alpha chars.   }
{-------------------------------------------------------------------------}

var
  XPos,YPos : integer;

begin
  get_abs_cursor(XPos,YPos);
  if XPos > 1 then
    begin
      if Screen(XPos - 1,YPos) in Alphabet then
        begin
          Previous := upcase(Screen(XPos - 1,YPos));
          if (XPos > 2) and ((Screen(XPos - 2,YPos) in Alphabet)
             or (Screen(XPos - 2,YPos) = ' ')) then
            MorePrevious := upcase(Screen(XPos - 2,YPos))
          else MorePrevious := ' ';
        end
      else
        begin
          Previous := ' ';
          MorePrevious := ' ';
        end;
    end
  else
    Previous := ' ';
end;


function COrd(ch : char) : integer;
{-------------------------------------------------------------------------}
{ Cord retorns the corrected ordinal value of the supplied character.     }
{                                                                         }
{ Cord actually returns the Alphabetic position of the character (A=1,B=2...) }
{ regardless of case. A non-alpha character returns the code 0.           }
{-------------------------------------------------------------------------}

begin
  if ch in Alphabet then
    COrd := ord(upcase(ch)) - ord('A') + 1
  else COrd := 0;
end;
```

```
procedure Menu(LocalKeys : RoundKey);
{-------------------------------------------------------------------------------}
{ Menu offers the user the help box of options and handles the response.       }
{-------------------------------------------------------------------------------}

var
  UpperCase,Done      : boolean;
  Choice,ExtendChoice : char;
  WStore              : array[1..25,1..160] of byte;
  RealX,RealY,i       : integer;
  HelpBox             : windowpointer;


procedure FillBox(Option: RoundKey);
{-------------------------------------------------------------------------------}
{ FillBox adjusts the Help box.                                                 }
{                                                                               }
{ The basic help box is filled out with the valid options around the selected }
{ key and the selected key itself.                                              }
{-------------------------------------------------------------------------------}

type
  CharString = string[80];
  UpdateStr = string[6];

var
  OptVal : integer;
  OptChars : array[1..3] of UpdateStr;
  PromptStr : CharString;
  PromptLength : byte absolute PromptStr;

begin {of fillbox}
  PromptStr := ' String : "' + MorePrevious + Previous + upcase(Keystroke) + '".
Hit menu option required';
  for Count := 1 to 6 do
    begin
      delete(HelpChars[KeyInsert[Count].y],KeyInsert[Count].x,6);
      if Option[Count] in Alphabet then
        insert('/ '+Option[Count]+' . ',HelpChars[KeyInsert[Count].y],
               KeyInsert[Count].x)
      else insert('-none-',HelpChars[KeyInsert[Count].y],KeyInsert[Count].x);
    end;
  HelpChars[KeyInsert[8].y][KeyInsert[8].x] := Keystroke;
  delete(HelpChars[KeyInsert[7].y],KeyInsert[7].x,PromptLength);
  insert(PromptStr,HelpChars[KeyInsert[7].y],KeyInsert[7].x)
end;
```

```
procedure MapChoice(Key1, Key2: char; Options : RoundKey);
{-------------------------------------------------------------------------------}
{ MapChoice reads the user response to the help box and maps it into a valid   }
{ character.                                                                    }
{                                                                               }
{ The response is evaluated for function key entry, ordinary key entry and the}
{ suitable return character (i.e. the character to be passed on to the next    }
{ keyboard handler in the chain) is determined. Debounce facilities are also   }
{ included in this routine.                                                     }
{-------------------------------------------------------------------------------}

var Accepted : boolean;


  procedure FuncKey(KeyStruck : char);
  {-----------------------------------------------------------------------------}
  { FuncKey handles the special case of extended scan code keys (<ESC> prefix  }
  { sequences) and takes suitable action. These are function and cursor keys.  }
  {-----------------------------------------------------------------------------}

  begin
    case KeyStruck of
      #59..#68     : ;                              { <F1> thru <F10> = leave }
      #71..#73,#75,
      #77,#79..#83 : Accepted := false;             { Cursor keys illegal now }
      else
        Accepted := false;
    end;
  end;


begin {of MapChoice}
  repeat
    begin
      Accepted := true;
      case upcase(Key1) of
        'Q','W','E','1','2','3' : if Options[1] in Alphabet then
                                    Keystroke := Options[1]
                                  else Accepted := false;
        'T','Y','U','5','6','7' : if Options[2] in Alphabet then
                                    Keystroke := Options[2]
                                  else Accepted := false;
        'O','P','@','9','0','-' : if Options[3] in Alphabet then
                                    Keystroke := Options[3]
                                  else Accepted := false;
        'A','S','D','\','Z','X' : if Options[4] in Alphabet then
                                    Keystroke := Options[4]
                                  else Accepted := false;
        'G','H','J','V','B','N' : if Options[5] in Alphabet then
                                    Keystroke := Options[5]
                                  else Accepted := false;
        'L',';',':',',','.','/' : if Options[6] in Alphabet then
                                    Keystroke := Options[6]
                                  else Accepted := false;
        ' ' : Keystroke := #0;
        #0  : FuncKey(Key2);
      else Accepted := false;
    end; {case}
    if not Accepted then
      begin
        sound(200);
        delay(100);
        nosound;
        Prompt[1] := ' Invalid keystroke.  Please try again ';
        OpenWindow(2,20,ord(Prompt[1][0]),1,Prompt,$4F,PromptWindow,Done);
        KeyStroke := #0;
        Key1 := readkey;
```

```
        if keypressed and (Key1 = #0) then Key2 := readkey else Key2 := #0;
        CloseWindow(PromptWindow);
      end;
    end;
  until Accepted;
end;


function DownCase(ch : char) : char;
{-------------------------------------------------------------------------}
{ DownCase returns the lower case equivalent of the supplied character.   }
{-------------------------------------------------------------------------}

begin
  if ch in ['A'..'Z'] then DownCase := chr(ord(ch) + 32)
  else DownCase := ch;
end;


begin {of menu}
  UpperCase := (upcase(Keystroke) = Keystroke);
  for i := 1 to 6 do if not UpperCase then LocalKeys[i] := DownCase(LocalKeys[i]);
  Get_Abs_Cursor(RealX,RealY);
  FillBox(LocalKeys);
  OpenWindow((80 - HelpSize.X) div 2,3,HelpSize.X,HelpSize.Y,
             HelpChars,HelpColor,HelpBox,Done);
  Choice := readkey;
  if (Choice = #0) and keypressed then ExtendChoice := readkey
  else ExtendChoice := #0;
  MapChoice(Choice,ExtendChoice,LocalKeys);
  CloseWindow(HelpBox);
  delay(300);   {scrap bounces and mistrikes}
  while keypressed do Dummy := readkey;
end;


function MaxVal(EarlyBegin_Trigram,Begin_Trigram : integer) : real;
{-------------------------------------------------------------------------}
{ MaxVal returns the maximum probability of occurrence achievable assuming the}
{ first two characters of the trigram.                                    }
{-------------------------------------------------------------------------}

var
  Maximum : real;
  AlphaLength : integer;

begin
  Maximum := 0;
  for AlphaLength := 1 to 26 do
    if Maximum < Matrix[EarlyBegin_Trigram,Begin_Trigram,AlphaLength] then
       Maximum := Matrix[EarlyBegin_Trigram,Begin_Trigram,AlphaLength];
  Maxval := Maximum / 5;
end;
```

```
procedure CheckUp(PreviousKey,LastKey : char);
{-------------------------------------------------------------------------------}
{ Checkup determines if the user requires assistance or not.                    }
{                                                                               }
{ The user is prompted with a single line window and a short tone from the      }
{ speaker. If requested, help will be offered, or else the window is closed     }
{ and normal activity takes place. (THIS CHARACTER IS LOST.)                    }
{-------------------------------------------------------------------------------}

var
  NearKeys : RoundKey;
  i,OldX,OldY : integer;
  ThisKey,Reply : char;

begin
  Blip;
  for i := 1 to 6 do
    if (matrix[COrd(moreprevious),COrd(previous),Keyboard[COrd(Keystroke),i]]
              < maxval(COrd(moreprevious),COrd(previous)))
      or(matrix[COrd(moreprevious),COrd(previous),Keyboard[COrd(Keystroke),i]]<1)
      then
        NearKeys[i] := ' '
      else
        NearKeys[i] := chr(Keyboard[COrd(Keystroke),i] - 1 + ord('A'));
  Prompt[1] := ' No match for "' + PreviousKey + LastKey + upcase(Keystroke)
               + '" found. <space> for help, or other key to continue. ';
  OpenWindow(2,22,ord(Prompt[1][0]),1,Prompt,$70,PromptWindow,Done);
  Reply := readkey;
  if keypressed then
    begin
      Reply := readkey;
      Reply := 'N';
    end;
  delay(200);
  while keypressed do Dummy := readkey;
  CloseWindow(PromptWindow);
  if Reply = ' ' then Menu(NearKeys);
end;

{$L KBDHAND.OBJ}
Procedure KbdHandler; external;
{-------------------------------------------------------------------------------}
{ KbdHandler is the keyboard handler.                                           }
{-------------------------------------------------------------------------------}


procedure PutInCSEG(OldVec : pointer); external;
{-------------------------------------------------------------------------------}
{ PutInCSEG places the supplied vector in a location in the CSEG for easy       }
{ access by the keyboard interrupt handler (KbdHandler).                        }
{-------------------------------------------------------------------------------}
```

```
Procedure TPKbdHand;
{-------------------------------------------------------------------------}
{ TPKbdHand is the part of the keyboard handler called by the assembler   }
{ program (KbdHandler). It performs high level functions required for window  }
{ functions and interface to the user.                                    }
{-------------------------------------------------------------------------}


begin
  Get_Abs_Cursor(OldX,OldY);
  restore_previous;
  if previous <> ' ' then
    if (matrix[cord(moreprevious),cord(previous),cord(keystroke)]
                < maxval(cord(moreprevious),cord(previous)))
        or (matrix[cord(moreprevious),cord(previous),cord(keystroke)] < 1)
          then
            checkup(moreprevious,previous);    { test validity of trigram }
  matrix[cord(moreprevious),cord(previous),cord(keystroke)] :=
            matrix[cord(moreprevious),cord(previous),cord(keystroke)] + 1;
  GotoXY(OldX,OldY);
  keyscan := ScanCode[ord(keystroke)];
end;

{$I M24KBD2.INC}

procedure Setup;
{-------------------------------------------------------------------------}
{ Setup performs initialisation using external files to load all the matrices }
{-------------------------------------------------------------------------}

begin
  ReadKeyFile(Keyboard,ScanCode);
  ReadLayout;
  ReadTrigrams;
end;


{The main program installs the new interrupt routine and makes it permanently
 resident as the keyboard interrupt.  The old keyboard interrupt is stored in
 OldInt16 so that it can be chained to or restored when TEACH is removed from
 memory. }

{----------M A I N    B L O C K-----------------------------------------}
{-------------------------------------------------------------------------}
{ The main block installs the keyboard handler vectors, initialises TEACH and }
{ terminates with the Terminate and Stay Resident call to DOS.            }
{-------------------------------------------------------------------------}

Begin                                   {**main**}

  Setup;

  Writeln(' Text Entry Analysis/Correction Helper loaded.
           Use <Alt-F10> to remove it.');

  GetIntVec($16,OldInt16);          { Store old interrupt vector   }
  PutInCSEG(OldInt16);             { Save in CSEG so it available }
  SwapVectors;
  SetIntVec($16,@KbdHandler);       { Install new vector           }

  Keep(0);                         { terminate and stay resident  }
end.
```

Appendix D - Text passages use in evaluating TEACH performance

Passage 1

ONCE THE ROLE OF DNA HAD BEEN IDENTIFIED THE INVESTIGATION OF ITS CHEMICAL MAKEUP BEGAN IN EARNEST IT WAS ALREADY KNOWN THAT DNA MOLECULES ARE LONG CHAINS OF ORGANIC COMPOUNDS CALLED NUCLEOTIDES AND THAT EACH NUCLEOTIDE LINK CONTAINS A SUGAR A PHOSPHATE AND ONE OF FOUR NITROGEN BASES ADENINE THYMINE GUANINE OR CYTOSINE COMMONLY SHORTENED TO A T G AND C IT STILL REMAINED THOUGH FOR SOMEONE TO DETERMINE HOW THESE SUBSTANCES COMBINE TO CREATE A DNA MOLECULE AND THIS TASK WAS TAKEN UP IN SEVERAL EUROPEAN AND NORTH AMERICAN LABORATORIES AT THE UNIVERSITY OF LONDON MAURICE WILKINS A STUDENT OF DNA WAS ATTEMPTING TO DETERMINE ITS MOLECULAR STRUCTURE BY THE PAINSTAKING TECHNIQUE OF X RAY CRYSTALLOGRAPHY X RAYS WERE PASSED THROUGH PURIFIED DNA CRYSTALS AND SCATTERED

Passage 2

AT THE HEARING GERALD HIS MOTHER OLDER BROTHER AND TWO PROBATION OFFICERS APPEARED BEFORE THE JUVENILE COURT JUDGE GERALDS FATHER WAS OUT OF TOWN ON BUSINESS THE COMPLAINING NEIGHBOUR DID NOT APPEAR NO ONE WAS SWORN AT THIS HEARING NO ONE MADE A TRANSCRIPT OR RECORDING OF THE PROCEEDINGS GERALD ANSWERED QUESTIONS BUT SOME CONFUSION REMAINED ABOUT WHAT HE SAID HIS MOTHER RECALLED THAT GERALD TOLD HER HE ONLY DIALLED MRS COOKS NUMBER AND THEN HANDED THE PHONE TO HIS FRIEND THE PROBATION OFFICER SAID GERALD ADMITTED MAKING THE CALL THE JUDGE SAID HE WOULD THINK ABOUT IT AND SENT GERALD BACK TO THE DETENTION HOME A FEW DAYS LATER HE RELEASED GERALD WITHOUT EXPLANATION THE SAME DAY HIS MOTHER RECEIVED NOTICE OF ANOTHER HEARING THERE IS NO RECORD OF THAT HEARING BUT IT ECHOED THE PREVIOUS ONE

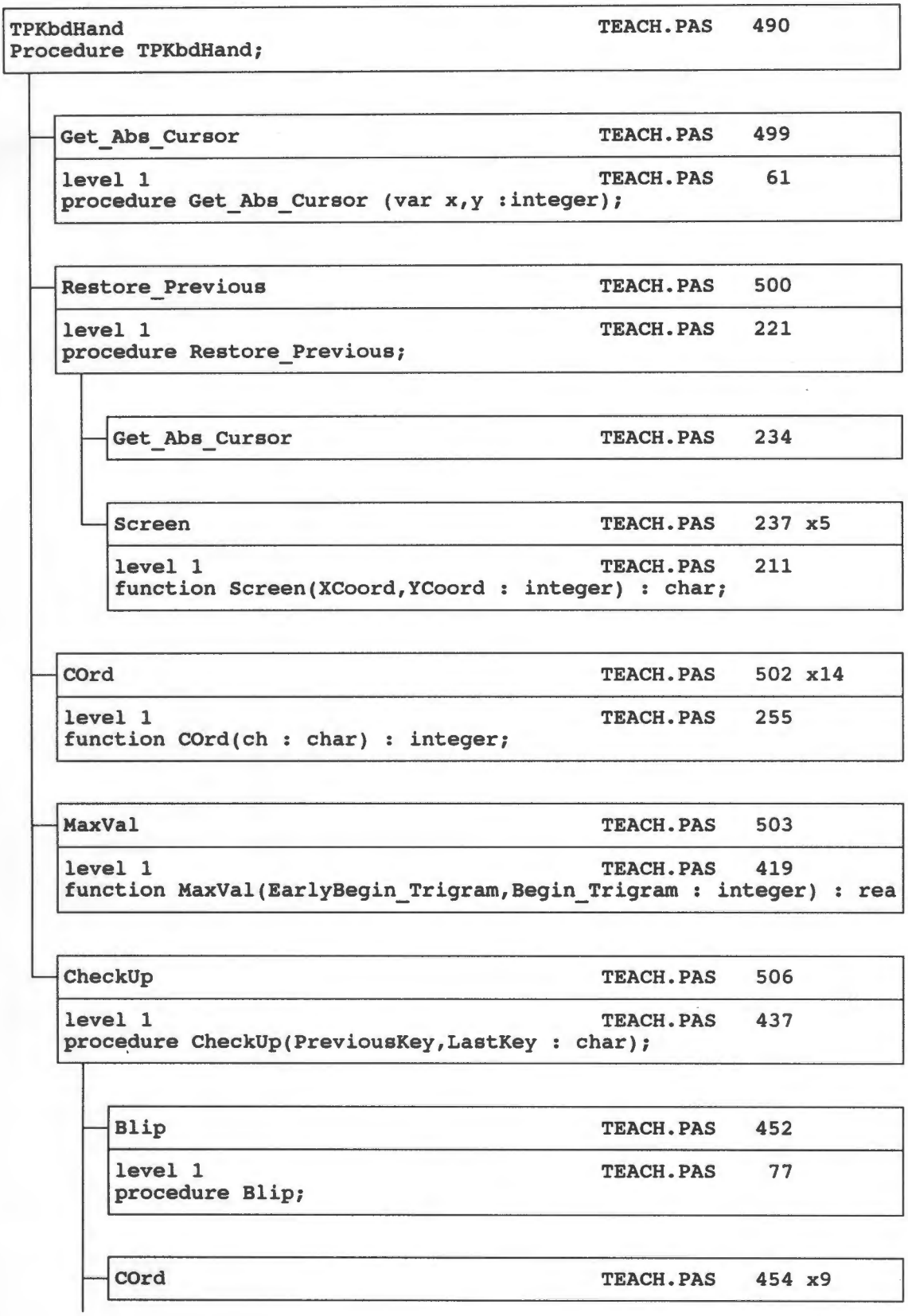
Passage 3

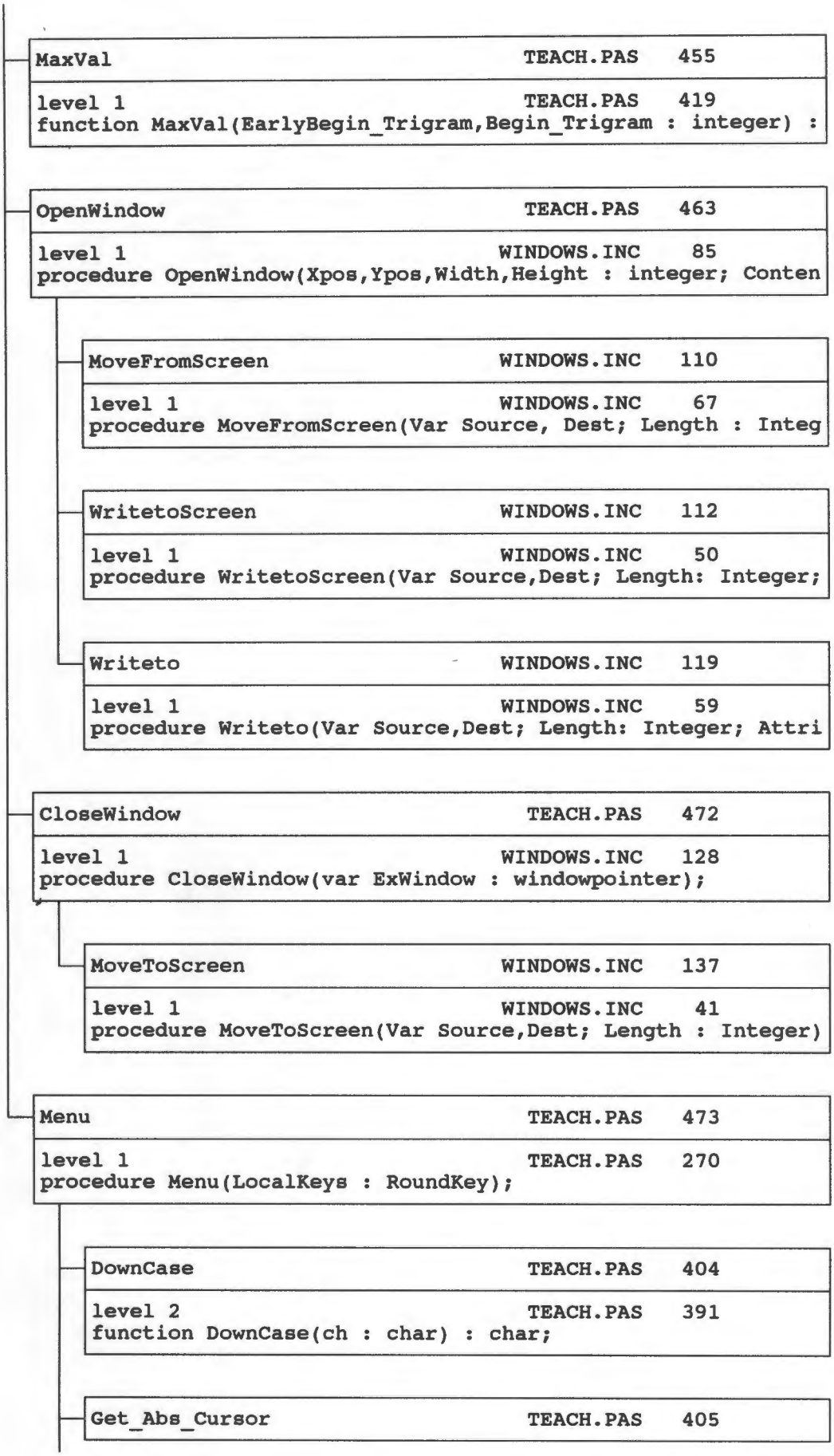
THERE IS THE SUSCEPTIBILITY OF THE POLITICAL ESTABLISHMENT TO THE EMOTIONS AND VAGARIES OF PUBLIC OPINION PARTICULARLY IN THIS DAY OF CONFUSING INTERACTION BETWEEN THE PUBLIC AND THE VARIOUS COMMERCIALIZED MASS MEDIA THERE IS THE INORDINATE INFLUENCE EXERCISED OVER AMERICAN FOREIGN POLICY BY INDIVIDUAL LOBBIES AND OTHER ORGANIZED MINORITIES AND THERE IS THE EXTRAORDINARY DIFFICULTY A DEMOCRATIC SOCIETY EXPERIENCES IN TAKING A BALANCED VIEW OF ANY OTHER COUNTRY THAT HAS ACQUIRED THE IMAGE OF A MILITARY AND POLITICAL ENEMY THE TENDENCY THAT IS TO DEHUMANIZE THAT IMAGE TO OVERSIMPLIFY IT TO IGNORE ITS COMPLEXITIES IN THE LIGHT OF THESE CONDITIONS I CAN WELL UNDERSTAND THAT DEALING WITH OUR GOVERNMENT CAN BE A FRUSTRATING EXPERIENCE AT TIMES FOR ANY FOREIGN REPRESENTATIVE

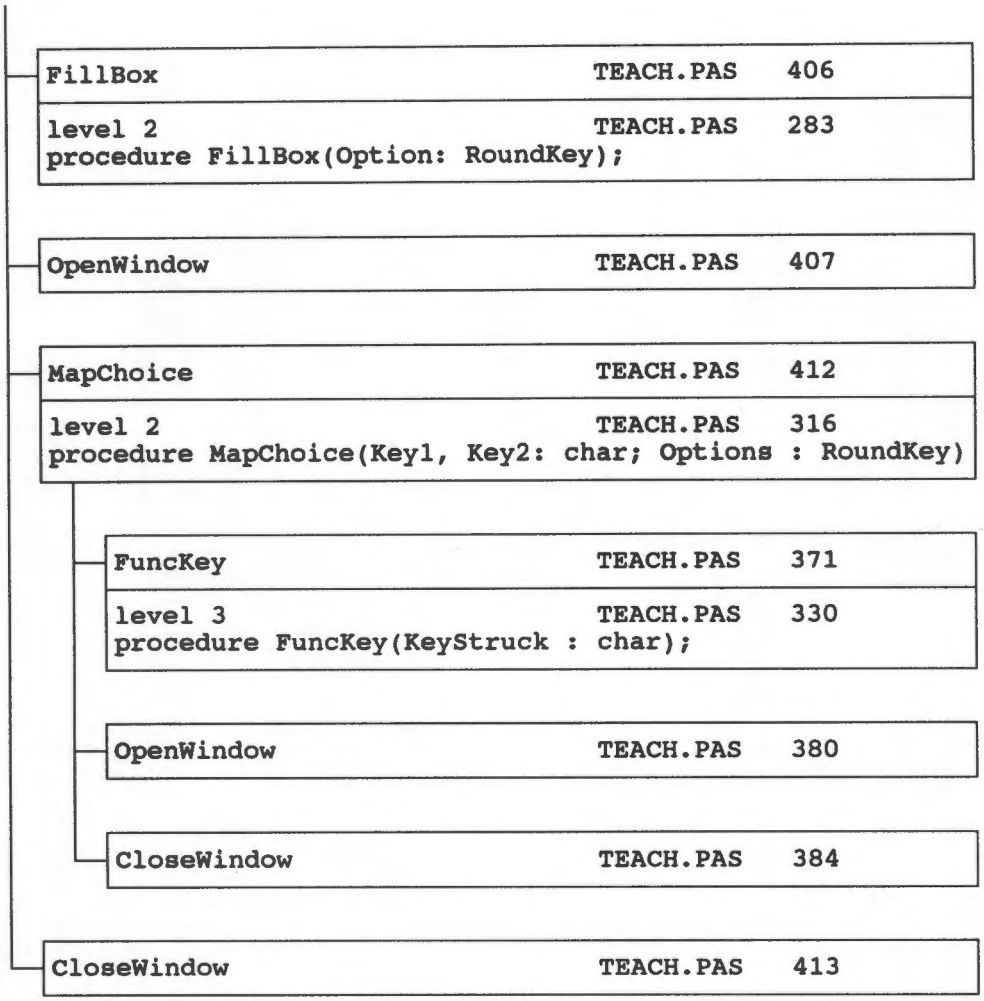
Passage 4

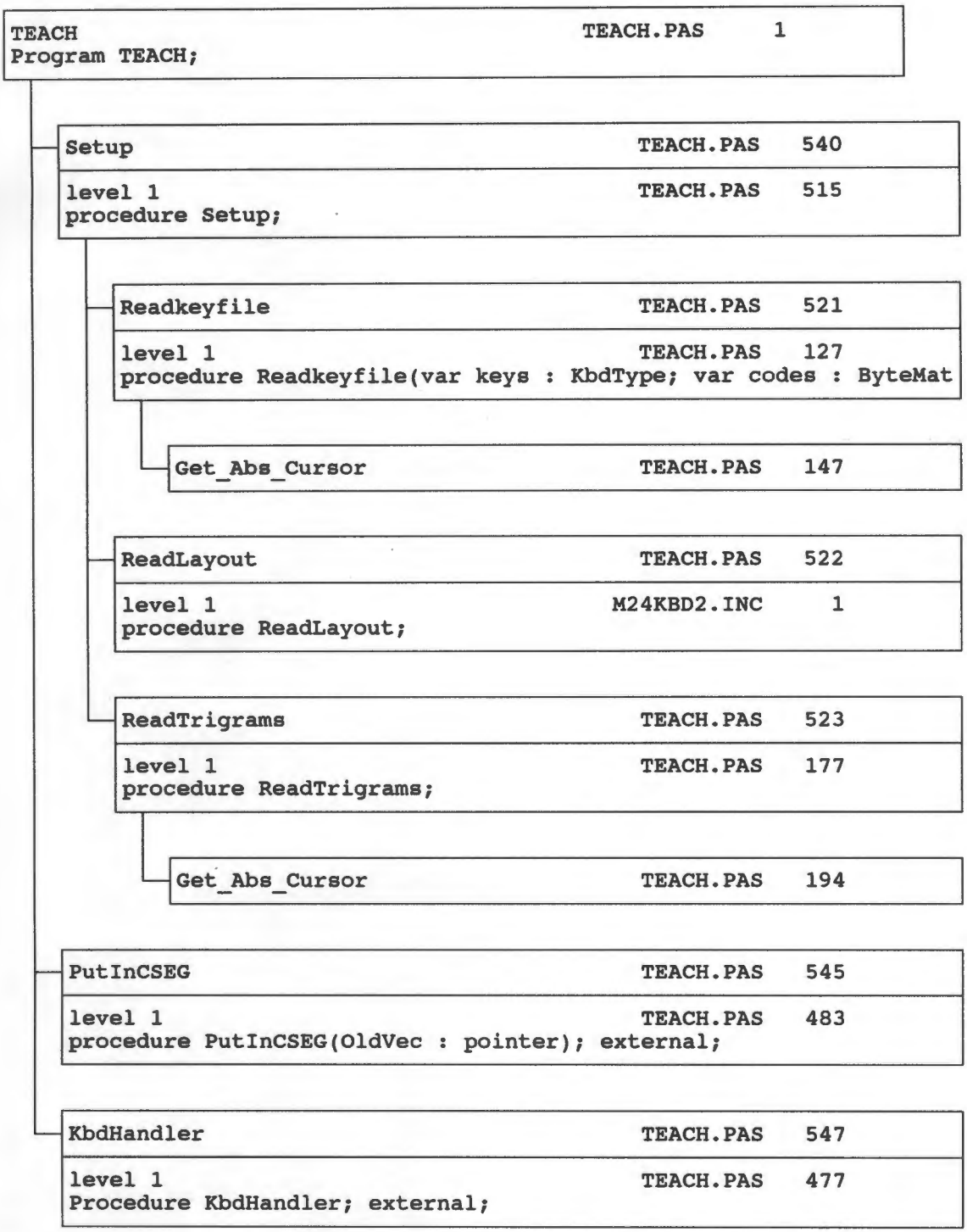
ACCOUNTABLE FOR MYSELF AS A QUADRIPLEGIC CONFINED TO A WHEELCHAIR I WAS ALWAYS GRATEFUL FOR VISITS TO A ZOO OR ART SHOW WITH A FAITHFUL FRIEND PUSHING ME FROM BEHIND HOWEVER I REMEMBER THAT I USUALLY CAME HOME WITH A STIFF NECK BECAUSE MY FRIENDS RARELY WOULD TURN THE CHAIR SO THAT I FACE THE EXHIBITS MY TOES WERE OFTEN BRUISED BECAUSE THE PUSHER MISJUDGED DOORWAYS AND OTHER OBJECTS AND I WAS FREQUENTLY ALL SLUMPED DOWN IN MY CHAIR FROM BEING PUSHED HEAD ON INTO BUMPS IN THE SIDEWALK FORTUNATELY THESE DISCOMFORTS ARE GONE FOREVER I STILL ENJOY EXCURSIONS WITH MY FRIENDS BUT THANKS TO MY PUFF AND SIP CONTROLLED MOTORIZED WHEELCHAIR I DO NOT HAVE TO RELY ON THEM FOR MOBILITY AND I CAN TURN THE CHAIR TO FACE WHATEVER DIRECTION I WISH I AM IN A BETTER POSITION TO SEE WHERE MY FEET ARE AND ALLOW FOR THEIR SAFETY IF AT ALL POSSIBLE I GO AROUND THE BUMPS IN THE SIDEWALK

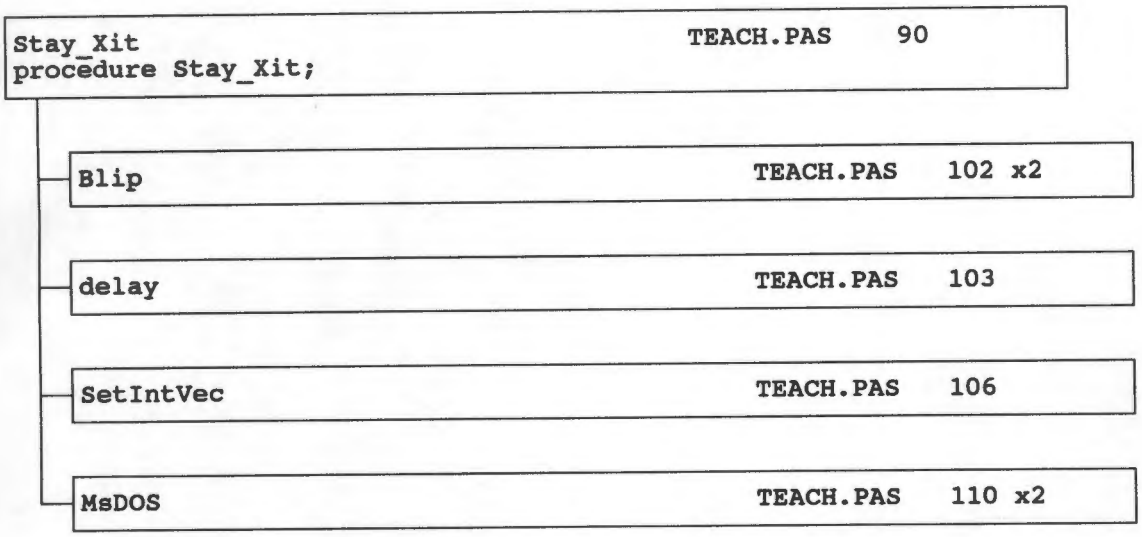
Appendix E - Tree diagram of TEACH source code

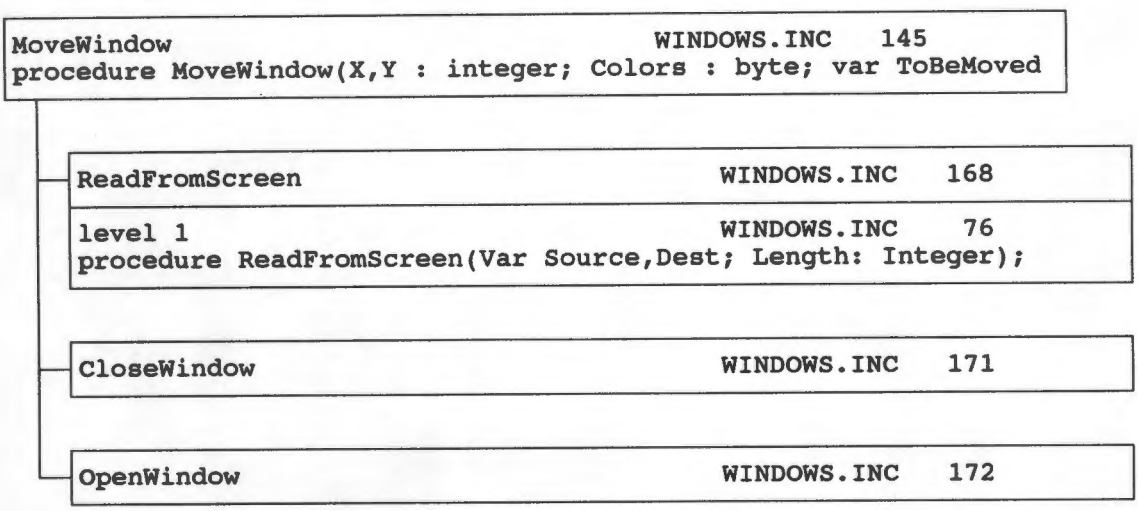



MaxVal TEACH.PAS 455
level 1 TEACH.PAS 419
function MaxVal(EarlyBegin_Trigram,Begin_Trigram : integer) :
OpenWindow TEACH.PAS 463
level 1 WINDOWS.INC 85
procedure OpenWindow(Xpos,Ypos,Width,Height : integer; Conten
MoveFromScreen WINDOWS.INC 110
level 1 WINDOWS.INC 67
procedure MoveFromScreen(Var Source, Dest; Length : Integ
WritetoScreen WINDOWS.INC 112
level 1 WINDOWS.INC 50
procedure WritetoScreen(Var Source,Dest; Length: Integer;
Writeto WINDOWS.INC 119
level 1 WINDOWS.INC 59
procedure Writeto(Var Source,Dest; Length: Integer; Attri
CloseWindow TEACH.PAS 472
level 1 WINDOWS.INC 128
procedure CloseWindow(var ExWindow : windowpointer);
MoveToScreen WINDOWS.INC 137
level 1 WINDOWS.INC 41
procedure MoveToScreen(Var Source,Dest; Length : Integer)
Menu TEACH.PAS 473
level 1 TEACH.PAS 270
procedure Menu(LocalKeys : RoundKey);
DownCase TEACH.PAS 404
level 2 TEACH.PAS 391
function DownCase(ch : char) : char;
Get_Abs_Cursor TEACH.PAS 405


FillBox TEACH.PAS 406
level 2 TEACH.PAS 283
procedure FillBox(Option: RoundKey);
OpenWindow TEACH.PAS 407
MapChoice TEACH.PAS 412
level 2 TEACH.PAS 316
procedure MapChoice(Key1, Key2: char; Options : RoundKey)
FuncKey TEACH.PAS 371
level 3 TEACH.PAS 330
procedure FuncKey(KeyStruck : char);
OpenWindow TEACH.PAS 380
CloseWindow TEACH.PAS 384
CloseWindow TEACH.PAS 413


TEACH TEACH.PAS 1
Program TEACH;
Setup TEACH.PAS 540
level 1 TEACH.PAS 515
procedure Setup;
Readkeyfile TEACH.PAS 521
level 1 TEACH.PAS 127
procedure Readkeyfile(var keys : KbdType; var codes : ByteMat
Get_Abs_Cursor TEACH.PAS 147
ReadLayout TEACH.PAS 522
level 1 M24KBD2.INC 1
procedure ReadLayout;
ReadTrigrams TEACH.PAS 523
level 1 TEACH.PAS 177
procedure ReadTrigrams;
Get_Abs_Cursor TEACH.PAS 194
PutInCSEG TEACH.PAS 545
level 1 TEACH.PAS 483
procedure PutInCSEG(OldVec : pointer); external;
KbdHandler TEACH.PAS 547
level 1 TEACH.PAS 477
Procedure KbdHandler; external;


Stay_Xit TEACH.PAS 90
procedure Stay_Xit;
Blip TEACH.PAS 102 x2
delay TEACH.PAS 103
SetIntVec TEACH.PAS 106
MsDOS TEACH.PAS 110 x2


MoveWindow WINDOWS.INC 145
procedure MoveWindow(X,Y : integer; Colors : byte; var ToBeMoved
ReadFromScreen WINDOWS.INC 168
level 1 WINDOWS.INC 76
procedure ReadFromScreen(Var Source,Dest; Length: Integer);
CloseWindow WINDOWS.INC 171
OpenWindow WINDOWS.INC 172