UNIVERSITY OF CAPE TOWN

IMPLEMENTING LINGUISTIC TEXT ANTICIPATION IN A WRITING DEVICE FOR THE DISABLED

Submitted to the Department of Biomedical Engineering
In partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE (MEDICINE)

by

ANNALU WALLER

October 1988

To

OUMA

whose faith in my abilities
always encouraged me to do my best.

ACKNOWLEDGEMENTS

I would like to thank the following people, without whom this thesis would not have materialised:

- To my parents and sisters for encouraging me to strive towards my ideals and for providing the love and support needed to attain them.

- To my supervisor, Dr David Boonzaier, for being my mentor and friend.

- To the staff and students of the Department of Biomedical Engineering for their constant encouragement, and to Menan du Plessis of the UCT Linguistics Department for her ideas and suggestions.

- To my family, friends and colleagues, for helping me to achieve my goals.

This research was supported by the Medical Research Council (MRC), UCT Postgraduate Scholarships and National Cash Register, South Africa.

INDEX

ABSTRACT

The advent of the microcomputer has provided the severely handicapped with the means to create text. Instead of using a keyboard, the disabled typist is able to scan and select linguistic items with an appropriate input switch.

The resulting communication rate is, however, prohibitively slow for writing and impractical for conversation. A variety of techniques is used to improve this rate and range from static letter matrices to more sophisticated methods in which words and phrases are anticipated. Although many anticipatory methods claim to be linguistically based, most, if not all, depend solely on letter and word frequency statistics.

A series of phonological rules can be used to anticipate the letter structure of most English words. This linguistically based system reflects a degree of "intelligence" not present in other anticipatory writing systems.

In order to evaluate and compare the new sytem with several existing techniques in practice, a programmable evaluation system has been developed on an IBM-compatible personal computer using the Artificial Intelligence language, LISP.

Different communication strategies are transcribed into rule-bases which serve as input to the software. The core program then executes the particular system under consideration. Input text can be processed in either manual or simulation mode and an

evaluation report is generated when the session ends.

The characteristics of efficient communication systems are introduced as a basis for this dissertation, after which the development and application of a linguistic anticipatory writing system is described. The design of the evaluation software is documented and the successful implementation of the various communication systems is discussed.

CHAPTER ONE

INTRODUCTION

The advent of the personal computer (PC) has totally changed the face of communication for the severely handicapped individual, for whom this activity was frustrating and sometimes even impossible. The availability of various applications-software packages has created new opportunities for disabled people in areas of conversation, education (e.g. word-processing and mathematics drill), recreation (e.g. chess and PacMan) and environmental control (e.g. turning appliances on/off and dialling/answering a telephone).

However, severe physical limitations often prevent conventional keyboard entry necessitating the use of alternative input techniques. Although, these techniques provide computer access, the typing rate is prohibitively slow.

Various attempts to design efficient input systems use statistical and redundancy characteristics of language. Savings of up to 60% of typing rates have been reported. Research into linguistics, discussed in this dissertation, has identified ways in which groups of letters can be anticipated.

In order to compare the performance of various input systems, an objective evaluation system is needed. A computer program has been written which simulates different input systems and compares them with one another.

The successful implementation of an effective communication system does not depend on the physical communication device alone. Factors such as assessment and training play an important role. These further considerations form the basis of an in-depth discussion of augmentative and alternative communication and constitutes the next three chapters. The fifth chapter introduces the linguistic basis for an anticipatory communication system and formulates the rules needed to implement such a system. This system is analyzed by the software and is compared with existing systems in chapter seven. The results of this analysis are discussed and recommendations for further research are made.

The major aims of this work can be summarised as follows:

- To develop a phonologically based rule structure to enable a scanning system to demonstrate some anticipation in selecting letters once the creation of a word has commenced.
- To develop an objective evaluation system to be used as a tool to compare various scanning techniques, including the one mentioned above, in terms of efficiency and redundancy as measured in quantitative terms as a benchmark procedure.

The purpose of this dissertation is therefore not to design a wordprocessor, but rather to develop a system which can be used to indicate ways in which computer programmers can design more efficient software for the physically disabled.

CHAPTER 2

COMMUNICATION SYSTEMS

INTRODUCTION

Communication is the process by which people formulate, send, receive and interpret ideas and concepts. The communication process involves a continual exchange of the roles of sender and receiver. In order to convey an idea, it must be transformed into a symbolic representation, e.g. words or pictures. These symbols are then transmitted using an appropriate transmission technique, e.g. speech or writing. The message is interpreted using auditory, visual and other sensory cues, and is then transformed (decoded) into an idea which can then be acted upon by the message receiver (figure 2-1).

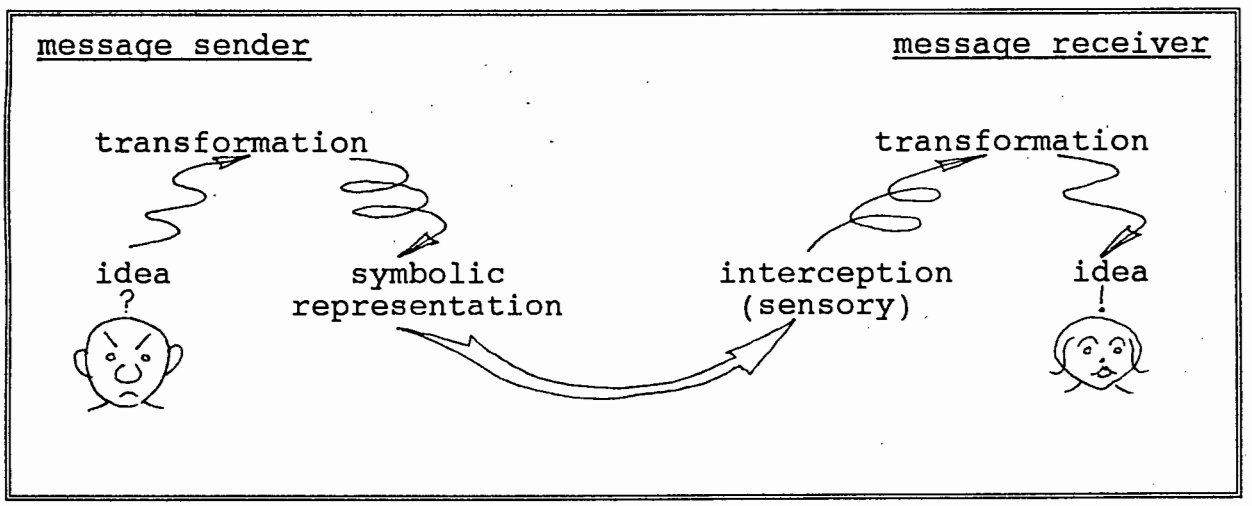


Fig: 2-1 The Communication Process

The transmission of messages involves both linguistic and non-linguistic behaviour. Linguistic behaviour includes speech and written language, whereas non-linguistic aspects involve body language and gestures.

To ensure effective communication, the message sender must be physically able to transmit a message. The message receiver needs the sensory ability to intercept the message. Both communication partners must be able to interpret the symbolic system if the message is to be correctly understood. A communication problem results if any component in the communication process is impaired.

For centuries, people with speech problems who were unable to communicate, used sign and/or written languages to interact with others. However, these augmentative communication systems are only successful when a) the disabled individual has the manual dexterity to sign, write or type, and b) the communication receiver can understand the message. Many physically disabled people who experience communication problems, are unable to use communication systems requiring good hand function. Augmentative and alternative communication (AAC) systems are thus needed to allow non-speaking people to assume the role of message sender (figure 2-2).

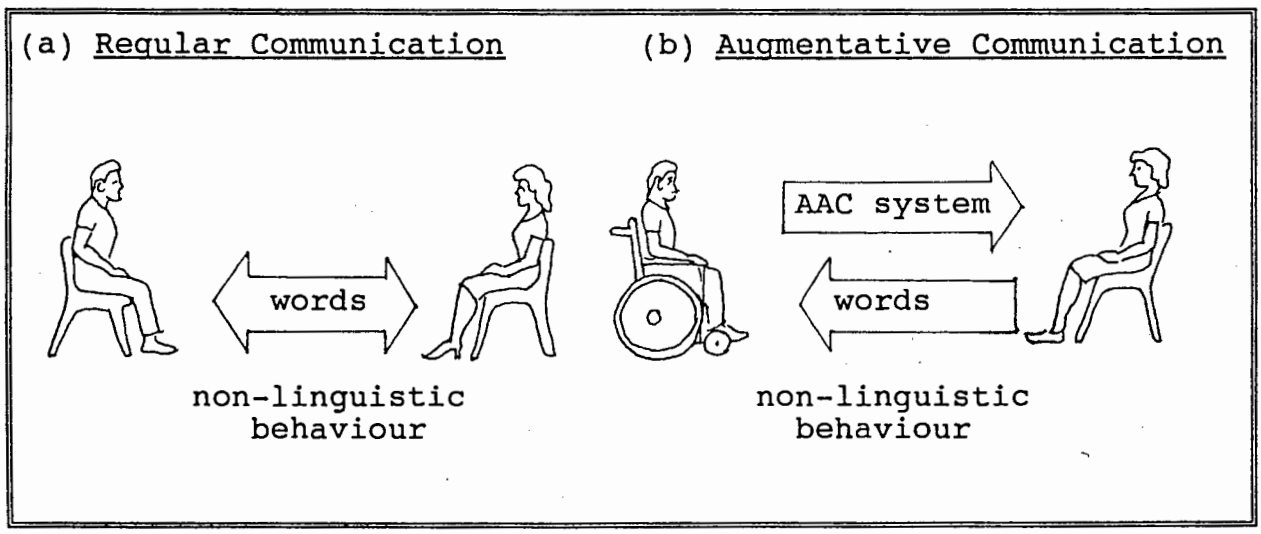


Fig: 2-2 The Communication Interaction

The philosophy of the use of AAC systems will be discussed as follows:

- An introduction to different communication problems;
- The identification of communication needs;
- Augmentative and alternative communication.

COMMUNICATION PROBLEMS

Communication can be divided into expressive and receptive communication. Expressive communication refers to the formulation, encoding and sending of messages whereas receptive communication refers to the interception, decoding and understanding of messages.

Impaired expressive communication indicates a difficulty in communication output (Creech, 1984). This may arise as a result of a motor disability affecting speech and/or physical movements - e.g. cerebral palsy, amytrophic lateral sclerosis (ALS) - or an inability to convert thoughts into words - e.g. expressive aphasia. Writing is another form of expressive communication and can be impeded by motor, visual and mental disabilities.

A loss of hearing, visual impairment, and the inability to understand language - e.g. receptive aphasia - can all lead to a receptive communication handicap.

A communication problem may be due to sensory, mental, psychological or physical (motor) impairments.

Sensory Impairment

An individual with a hearing impairment experiences difficulty in both receptive (hearing) and expressive (spoken) communication. A visually disabled person also has difficulty with both expressive and receptive forms of communication. The person with a hearing loss, however, has a conversational impairment whereas the visually disabled person cannot read or write.

Various communication systems enable the deaf and the blind to overcome their communication problems. Unaided systems, e.g. sign language, provide conversation for the deaf (Vanderheiden & Lloyd, 1986), while braille provides an aided written communication system for the blind (Arditi & Gillman, 1986). Development of technology provides more and more sophisticated systems such as film subtitling, telecommunication devices and cochlear implants for the deaf (Damper et al, 1979; Hoyt, 1985; Loomis et al, 1983; Minneman, 1984) and braille input-output (I/O) devices for the blind (Grossner et al, 1983; Fant, 1982; Stoffel, 1982).

Mental Impairment

The delay in the development of speech and motor control affects the communication of the mentally retarded individual. Gesture, sign and symbolic communication languages, are used to provide conversation for individuals with language and motor delays (Meyers, 1983).

Microcomputers can be used to provide a slow, repetitive medium for teaching, as well as providing a means to simulate real-world situations, through the use of graphics and speech synthesis (Young, 1983).

Psychological Impairment

Communication handicaps can result from a psychological disability. Autism is one such disability in which communication can be severely impaired. Sign language and graphic communication boards are used to facilitate communication when the autistic individual is reluctant to engage in direct communication (Scrimshaw). The microcomputer has proved a motivating factor in breaking down this barrier,by providing a non-threatening environment for social interaction (Papert, 1980).

Physical Impairment

Physical impairments often result in limited communication when the motor control of oral and limb function is impaired. Physical disfigurement, paralysis, spasticity and uncontrolled movements can impede spoken and/or written communication. Handicapping conditions caused by neurological damage (e.g. spinal cord injury, cerebral palsy, and cerebrovascular accidents (CVA - stroke)), degenerative conditions (e.g. amytrophic lateral sclerosis and muscular dystrophy), and physical trauma (e.g. vocal chord damage). All have devastating effects on communication.

In many cases conventional augmentative communication systems are appropriate. Writing and sign language can be used when motor abilities allow adequate hand function. However, when severe motor handicaps exist, alternative systems have to be employed.

Severe physical impairment often manifests in multiple handicaps which might include a number of the impairments mentioned above. This dissertation addresses those communication handicaps which result primarily from physical impairments. Unless otherwise indicated, terms relating to disability will refer to physical disability.

COMMUNICATION NEEDS

Communication is fundamental to interpersonal interaction and occurs in response to several communication needs (Vanderheiden, 1983; Vanderheiden & Lloyd, 1986). These needs can be summarized as follows:

The **communication of basic needs** necessitates the transfer of essential information in a quick and easy and way which can be readily understood (MacDonald, 1980; McNaughton, 1985).

Conversation also depends on a sufficiently fast rate of communication to facilitate functional interaction. Any restriction on the communication rate diminishes the quality of conversation (Vanderheiden, 1983).

Writing and messaging describe permanent forms of communication. An ability to create text is essential for school work and occupational requirements (Vanderheiden, 1983). The speed at which written text is produced, is not as crucial as in the communication of basic needs and conversation, as it does not primarily involve human interaction. However, it is still an important consideration as more rapid machine communication can contribute towards greater efficiency in ergonomic terms.

Drawing allows for perceptual, psychological and recreational development by providing creative exploration, allowing expression of feelings and providing pictorial and graphic representation of facts and ideas (Papert, 1980; Weir et al., 1982).

The emergence of the computer age has created new communication needs and opportunities. Electronic communication in the form of bulletin boards, data banks, information retrieval, electronic payments, and work opportunities (Pilgrim), all require some type of computer access.

Environmental control systems are to be found in any home, e.g. remote controls used to change television channels and to open electric garage doors. The availability of these systems has provided the severely disabled with the means to control their own environment for the first time (Boonzaier & Kleviansky, 1987; Demasco & Horstmann, 1987; Leifer, 1981). In order to benefit from environmental control systems, access to computer equipment is essential.

AUGMENTATIVE AND ALTERNATIVE COMMUNICATION

An augmentative communication system describes a system which enhances an existing communication system. Body language and gestures would constitute an augmentative communication system for an individual with a speech impediment. An alternative system refers to a substitute for an impaired communication system. A typewriter, would constitute an alternative communication system for someone who is physically unable to write.

Various forms of augmentative and alternative communication (AAC) have always been used. Normal verbal conversation would not be complete without the non-linguistic forms of communication which augment the dialogue. Gestures and body language form part of everyone's communication repertoire, as do changes in cadence, emphasis, etc. Printing, records, tape-recorders and other forms of communication storage augment our ability to speak over long distances. They also allow us to read or listen repetitively. Smoke signals, tom-tom drums, public-address systems and telecommunications provide augmentative communication systems, accepted as natural ways of projecting communication further than the human voice can carry. Spectacles augment the sight of people with impaired vision, but are no longer seen as extraordinary, and are an accepted part of the individual wearing them. Artistic representations, dress and music are also used to communicate feelings and ideas.

What of alternative communication? The written word provides a permanent record using traditional orthography [1] as a means of recording and communicating a message. Typewriters and wordprocesssors are used as alternatives to a pen or pencil.

Fortunately, these methods of providing AAC are seen as complementary to our normal communicative abilities. It is important for people - the general public as well as handicapped individuals - to become more aware of, and to learn to understand the possibilities offered by the various forms of AAC available to the disabled.

Individuals who are unable to use conventional means for communication rely on AAC systems. These systems should ideally not affect the interaction between non-speaking individuals and their communicating partners.

Various communication systems have been developed for disabled persons. Communication symbols or symbol sets refer to the format by which meaning is conveyed. Symbol sets include body signals, gross gestures, signs, symbolic pictures, and traditional orthography. A symbol system refers to the way in which the symbols of a symbol set can be structured to convey a meaningful message. A communication device refers to the physical implementation of a symbol system and can range from "low-tech", e.g. pen and paper, to "high-tech", e.g. a computerized system.

1. Derived from ortho Greek for correct and -graphia Greek for styles of writing i.e. correct or conventional spelling.

Classification Taxonomies

Currently, two classification systems are used to distinguish different AAC symbols and symbol systems. Lloyd and Karlan (1984), and Lloyd and Fuller (1986), use the more accepted method which classifies symbols and symbol sets as aided or unaided.

Unaided systems are those communication systems in which symbols are formed by using parts of the body, e.g. voice, hands and limbs. In contrast, **aided** systems are those in which the formation and manipulation of symbols and symbol sets require additional devices, e.g. pencil, boards and computers. Table 2-3 classifies several communication systems into aided and unaided systems. .

Unaided	Aided
Body language	Electronic devices
Voicing	Communication boards
Eyegaze	Typewriters
Gestures	Mechanical devices
Pointing	Orthography
Signing	
Speech	

Table: 2-3 Unaided and aided classification.

The second classification method, described by Vanderheiden and Lloyd (1986), distinguishes between **dynamic** and **static** symbols and symbol sets. A dynamic symbol is one which relies on movement to convey meaning, while static symbols do not need movement. However, an ambiguity arises when a symbol system contains both dynamic and static symbols. For instance, most symbols in American Sign Language (ASL) use movement to convey meaning and are thus regarded as dynamic. Some signs, however, do not depend on movement and are static (e.g. "home", "time").

The aided/unaided taxonomy is unambiguous and will be used in this dissertation.

TOWARDS AN EFFICIENT COMMUNICATION SYSTEM

The ultimate goal facing AAC professionals is to make the communication system as transparent as possible to both communicating partners. A transparent system should encourage direct interaction with the user. When conversing with someone who uses an interpreter, for example, it is very easy to ignore the disabled partner and to interact largely with the interpreter. The communication system should integrate into the user's person, thus reducing the observable disability.

Advances in microcomputer technology have provided the means by which any physically disabled individual can access technology. Communication devices are becoming far more compact and portable, and are thus aesthetically pleasing and socially acceptable.

These factors are superfluous however, if the resulting communication is strained and stilted. Several factors contribute to the effectiveness of a communication system. Vanderheiden and Lloyd (1986) divide these factors into three main areas:

- the functionality of the system and its ability to meet communication needs;
- the availability and usability of the system; and
- the acceptability and compatibility of the system within the environment.

Functionality / Ability to meet communication needs

First, a communication system has to be functional and yet still meet the communicator's communicative needs. The requirements necessary for the communication of basic needs differ from those necessary for conversation, which are again different from writing. The main factor under consideration here is the rate (speed) at which information transfer occurs. The usual communication rate of an individual of approximately 180 words per minute (wpm), compares unfavourably with the communication rate of 2-8 wpm of a non-speaking individual who uses a scanning technique to spell out words (Vanderheiden, 1983).

This rate of communication is not functional when someone needs to express vital information as quickly as possible. It is also not viable to attempt to converse if one of the communicating partners is interacting at a rate which is more than 25 times slower than the other! Although communication can occur in such circumstances, it cannot be deemed as conversation in the true sense of the word. The resulting interaction between the communicating partners is invariably curtailed, one-sided and highly frustrating for both partners.

Writing can be achieved using typewriters, wordprocessors or adapted writing systems. In terms of writing speed, the rate of 2-8 wpm is approximately five times slower than the usual 30-35 wpm for regular typists (Vanderheiden, 1983).

Other factors that contribute to the functionality of a communication system include:

- openness;
- assertability;
- projection;
- display permanence;
- correctibility and
- expandability.

Openness refers to the degree of fluency which characterizes a system. An orthographic system is open as any word can be expressed by spelling it, letter by letter. A vocabulary-based system, be it word-based or picture-based, is closed or constrained as a limited vocabulary is available. The more open a system is, the slower the communication rate. Openness and speed are thus dependent on one other.

The ability to assert oneself in a conversation is a characteristic often neglected in communication systems. The augmentative communicator requires the ability to initiate, control and terminate communication. Examples of appropriate conversational utterances should include phrases such as: "I have something to say!"; "I'd like to participate in the conversation!"; "Can we change the subject, please?"; "I have said all I want to say!"; or "Goodbye!".

Vocalization, body movements, electrical buzzers and voice synthesizers enable the communicator to **project communication** onto a group of people or to attract attention in order to initiate,

control and terminate a conversation.

Display permanence simplifies the listener's task of collating individual symbols into meaningful sentences. Many electronic devices update a display from which an entire message can be read. However, manual and unaided systems expect the listener to keep track of the conversation. Selection of letters to form words by the "speaker" can impose great cognitive loads on the "listener". Although display permanence is important, hard copies of messages are not always advisable, as in instances where confidentiality is important to the communicating partners.

As a result of the prohibitive communication rate, the speaking partner subconsciously attempts to attain normal communication rates by anticipating what the non-speaking partner is about to say. Unfortunately, the non-speaking partner learns to accept the anticipated symbol, even if it was not the original target, to ensure the continuation of the conversation. **Correctability** is therefore an essential factor in a functional communication system. The non-speaking partner must be able to indicate a wrong selection, and must be encouraged to correct mistakes. Speaking partners must be trained to resist anticipation unless it forms part of the communication system (in which case the non-speaker must be able to indicate whether or not an anticipation is correct). It is easier to implement and use a correction feature on an electronic writing device as the user is in full control. The lack of time constraints make it easier for the user to correct typing mistakes. Caution must be taken when an anticipatory system is implemented, to ensure that the user is

given an opportunity to accept or refuse an anticipation.

Examinations pose a unique problem for disabled students. Extensions of time do not always compensate for the slower rate of communication as added stress factors slow down the communication rate still further and fatigue is increased with time. Oral examinations are fraught with problems as it is very difficult to follow a train of thought without a written record, or to retrace steps in order to correct errors. In an attempt to simulate an actual written examination, an amanuensis can be provided [2]. By speaking, or using an augmentative communication system, the candidate conveys to the writer what is required. Although this procedure is fairly successful, it is not always possible to communicate the desired information, with all its nuances, accurately. When dealing with symbolic subjects, such as mathematics, this is even more complex. An efficient writing system, combining the positive aspects of independence and amanuensis, is thus very important for students.

Communication systems must have the potential for expansion. A conversational system which is introduced when the communicator is young should expand along with the user's communication development. As communication needs change, so should the communication system.

Technology is developing continuously. In order to utilize new technology, a communication device has to be expandable.

2. An amanensis describes someone who writes from dictation. Derived from (servus) a manu Latin for serve as a secretary and ensis Latin for belonging to

Financial considerations often dictate that equipment be purchased on a piecemeal basis. Unless a device can be added to in this way, the initial cost of a total system may be prohibitive. Further, the advantages of new peripheral devices - e.g. printers, keyboards, speech input/output devices - and software can only be harnessed if the basic hardware has the ability to expand and adapt.

Availability / Usability

Any communication system must be both available and usable if the system is to be successful. Factors which contribute to this include:

- portability;
- position independence;
- independence;
- intelligibility and obviousness;
- durability;
- total cost;
- appropriateness.

Communication is necessary in any and every situation. Although writing can be done at a stationary workstation, a portable communication system is essential for conversation. Conversation is used in all types of situations and has to be readily available all the time.

An unaided system is the most portable of systems as the definition implies. Communication boards and an increasing number of electronic communication devices are relatively portable - e.g. portable typewriters and electronic notepads.

In order to access a communication system efficiently, it is often necessary to seat the disabled individual in a position which facilitates controlled movements. However, it is unnatural to insist that the individual remain in a structured position at all times. More dynamic and effective use of a variety of positions is needed. During play, outings, relaxation, bath time, etc., the individual may not be in an optimum position, and yet still need to communicate.

The augmentative communicator should be as independent as possible using his communication system. Many non-speaking individuals rely on a helper to translate their utterances for the listener/s. This dependence on another person poses problems when the helper is not available - the communicator is then unable to converse. Even when the helper is available, a conflict may occur if the communicator wishes his utterances to be confidential.

It is inefficient and demotivating to call on a helper to switch an electronic aid on or off, or to request that a communication system be set up so that basic communication can take place. Non-speaking people tend to forgo communication if the involved setting-up procedure is too cumbersome.

The level of independence which a user can have also relates to the intelligibility and obviousness of the communication system.

The more complex a system, the less likelihood of a wide circle of communicating partners. Unless the speaking partner is comfortable using the system, communication will be inhibited and fail. If a system is logical, undemanding and easy to use, there will be a greater chance for communication interaction with strangers and those unfamiliar with the system. Sign language or a sophisticated coding system may be very efficient, but unless the communicating partners are fluent in the system, communication will be difficult and break down.

A communication system should form an integral part of an augmentative communicator's lifestyle. If the communication system is not reliable and consistent, it will end up not being used. **Durability** concerns all types of aided systems. For instance, a communication board should be available during bath- and meal-times. Technical aids are likely to be dropped or banged every so often and should be built to reasonably withstand occasional mishaps. Unaided communication systems are also likely to "degrade" if the physical abilities of the user deteriorate. Although this may be unavoidable if the disability is degenerative in nature, physiotherapy may improve or stabilize functions required for the effective communication. Whether a communication system is aided or unaided, "service" and "backup" should be considered.

The cost of a communication system, especially an electronic device, is often seen to be prohibitive and is used as the reason for dismissing such an acquisition. When the cost of a system is calculated, it is also necessary to calculate the broader life-

cycle savings which could arise if the equipment was used. Without an efficient system, the non-speaking individual has to rely on a dedicated helper, who often has to be paid. Without this helper, communication is difficult and even impossible. The non-speaker is thus totally dependent on a full-time staff member. If an efficient system is supplied, there is more scope for independence, allowing helpers to distribute services between a number of disabled people and a variety of essential chores, thereby lowering the costs. The acquisition of a high-technology device will not only provide communication for the user, but function as an occupational tool as well and thus further reduce the need for a helper. For the severely disabled, the microcomputer has provided a unique occupational tool and many people, who previously were not able to contribute to society, are now involved in computer-based employment.

The appropriateness of a communication system is closely related to all of the points considered above. The variety of available computer systems is enormous, and unless a carefully matched system is employed, it will be frustrating, and may never reach its potential.

Acceptability / compatibility with environment

It is useless to hope that a communication system will be effective if it is not acceptable in the communicator's environment. In order to communicate, both communicating partners have to understand, and be comfortable with, the communication

system. Various factors determine the acceptability and compatibility of a system within an environment. Included in these factors are:

- aesthetic appeal;
- materials and practice compatibility;
- training issues;
- adaptability;
- computer compatibility; and
- inter-system/device compatibility.

Aesthetic appeal is ergonomically very important. A rehabilitation device should blend into the user's environment and not be seen as a separate entity, but rather as part of the user. When communicating with a non-speaking person, there is often a tendency to disregard the disabled person by "talking" to the communication system. The system should be transparent enough to combat this tendency and encourage direct interaction with the augmentative communicator.

Communication systems can be implemented by using a variety of materials. In lower income groups, it may not be possible to purchase and maintain sophisticated electronic devices. In this case, it is more feasible to implement the communication system on readily available materials, e.g. paper or cardboard. Although this may be thought to be "primitive", it will probably be the most efficient and appropriate way of implementing a system.

Training issues must be dealt with at an early stage of implementation. These can be viewed on two levels. Firstly,

professionals have to be familiar with the concept of introducing AAC before it can be successful (Hill et al, 1987): Speech and hearing therapists in South Africa have attempted to introduce Blissymbolics on numerous occasions in the past. These attempts were relatively unsuccessful until formal training courses were established. The first three-day course was held in Cape Town at the beginning on 1987 (Waller, 1987). A further eight courses have been held throughout South Africa. The course covers all aspects of AAC and is based on the "Blissymbolics Independent Study Guide" (MacNaughton, 1985). Although the primary focus of the course is on Blissymbolics, much of the time is spent on the assessment and application of AAC systems as a whole. Participants are encouraged to take an active part in the workshops and worksheets and role play are used extensively. The introduction of the training courses has resulted in a greater awareness of AAC in South Africa and AAC programmes are being implemented successfully in a number of schools and institutions. A total of 135 participants have been trained to date and represent a variety of disciplines (fig. 2-4):

Clinical Psychologists	2
Graphic Artist/Author	1
Logopaedic Students (final year)	18
Medical Practitioner	1
Occcupational Therapists	22
Parents	2
Physiotherapists	8
Social Worker	1
Speech and Hearing Therapists	31
Teachers	49
Total	135

Fig: 2-4 Blissymbolics Training Course Participants for the period 1987 - 1988.

It is also essential that the augmentative user be taught to use his/her system efficiently and effectively (Barker, 1987). In order to use a communication system, the user may have to follow an intensive training course during which fundamental perceptual skills are learnt. Instead of training the perspective user, an AAC system is often presented in its entirety. Because the user is initially unable to use the system, it is discarded as being inappropriate and useless. This scenario often occurs when a child is given a symbol board, with 200 to 500 symbols on it, and is expected to use it immediately. The child should rather be introduced to the symbols progressively while experiencing the concept of active communication. Only when the child understands the use and function of a communication system will he/she be able to use the system appropriately (MacDonald, 1982).

The fact that no disabled individual presents with exactly the same pattern of abilities and disabilities as another, causes great difficulties when choosing a communication device. The implementation of a manual system is time consuming and such systems are difficult to modify and expand. A communication board is constructed for an individual person and is hence appropriate for that person's needs. As the individual's communication needs change, so should the communication system. The more adaptable the system, the more applicable. The adaptability of a system is especially important when introducing a system to a young child. A basic system which lends itself to easy modification will encourage participation by the child.

An electronic communication system is more easily adapted, in terms of both input selection mode and vocabulary selection. A system which is unable to change, would impede the effective functioning of the device. However, a system in which a selection of input techniques, switch allocations, and delay times can be offered, would prove more useful. For instance, an electronic device could be used more efficiently if the system's characteristics could change each time a new user used the device or when a user's characteristics changed. The vocabulary should also be programmable if the device is to be generally useful.

An important function of the re/habiltation [3] process is to enable the disabled person to participate in society in a meaningful way. For the augmentative communicator, this participation is only possible if the communicating partners are able to interpret the messages transmitted by the message sender. A communication system using spoken and/or written language is most appropriate as society is familiar with these output modes.

When spoken or written output is not possible, it is essential that there is a similarity with conventional systems. When a symbolic system is used, written words above individual symbols enable the man in the street to understand the communication system. Individuals using sign language, or those having speech impediments, should have augmentative strategies which are

3. The term "habilitation" (Latin: habilis- have, use) is used when working with people who have been disabled at birth, or at a young age. "Rehabilitation" refers to work with people who have lost the ability to do some activity due to a disability acquired later on in life.

understood by people who do not have a knowledge of signing or who have difficulty in understanding uncoordinated speech. A notebook and pencil or an introduction note can mean the difference between successful and unsuccessful communication (figure 2-5).

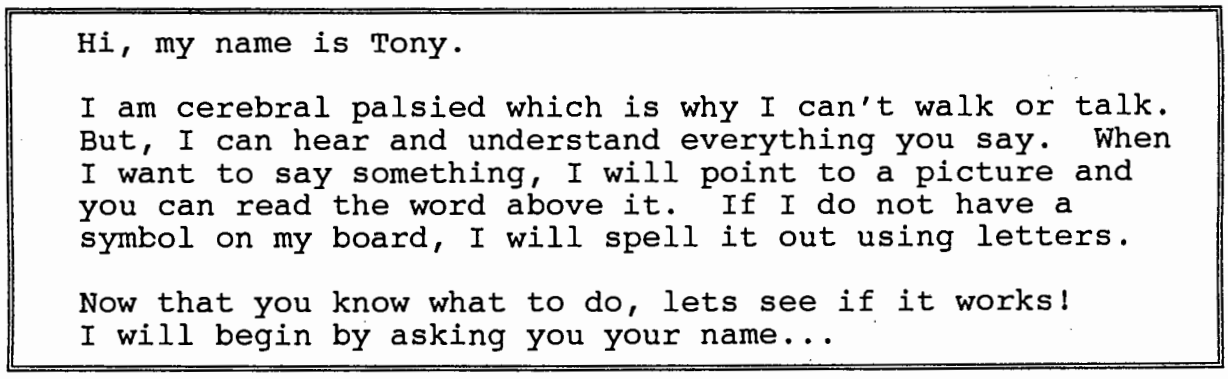


Fig: 2-5 An introduction note on a symbol board.

The need for compatible systems is very important in an educational environment. Although an augmentative system may be unconventional, there is no reason why the user should not be encouraged to master a conventional system, e.g. reading and writing. This learning experience is simplified if the communication system is externally consistent with conventional systems.

Augmentative communication systems are designed to enhance other communication systems, each being compatible with the other. A communication board should not hinder the use of gestures and sign language. There is also a need for **inter-device compatibility** where communication devices can interface with additional equipment. A communication device might double as an input device to a computer, thus allowing the user to interface with a commercially available computer.

Computer compatibility is another essential factor when choosing an electronic communication device. This is especially true when the communication device is to function as a writing and/or employment related tool. It is advisable to recommend a personal computer which is compatible with the make being used in the school or work environment. One of the biggest advantages of using the microcomputer as a rehabilitation tool is its ability to equalize the disabled and able-bodied individual. This advantage would be nullified if the disabled person could not operate a computer at work because of poor foresight and planning on the part of the rehabilitation consultant.

CHAPTER THREE

COMMUNICATION DEVICES

INTERFACING

Physically disabled individuals with communication problems often have difficulties when accessing communication devices: Pen, paper and typewriters, are effective communication devices for someone who is unable to speak. However, such devices require a fair degree of physical dexterity.

When dealing with severely disabled individuals, it is necessary to be aware of the handicap that the impairment imposes on the individual. When an able-bodied user interacts with a typewriter or a computer system, the keyboard is the standard means of input. However, it may be physically impossible for a disabled individual to use a conventional keyboard.

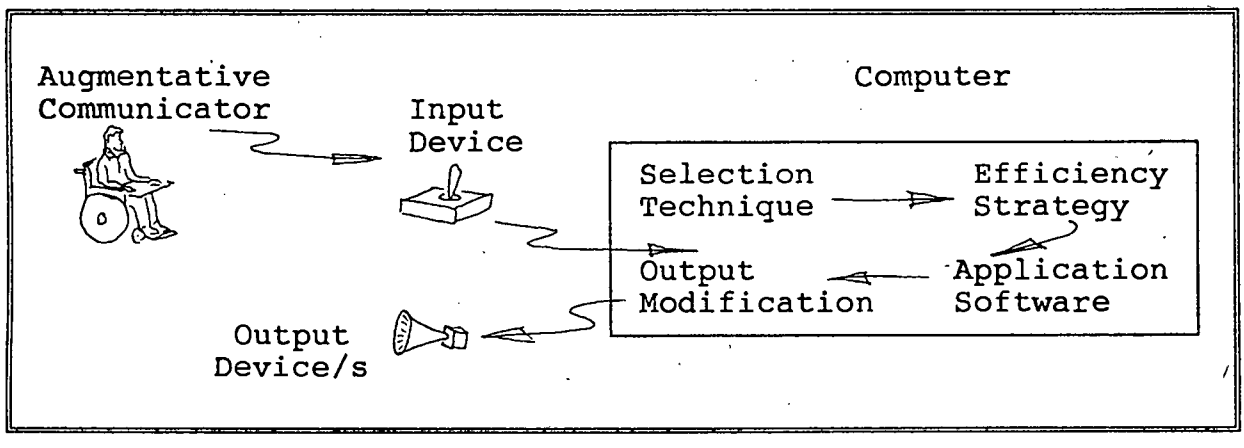


Fig: 3-1 The Computer Interface

In order to provide a physically disabled person with a means of communication, a computer-based device can be used. Figure 3-1 illustrates the different interfaces within a computer system. The input method allows physical access to the computer, e.g. the

keyboard. The selection technique describes the method by which symbols are chosen from the symbol set and made available to the computer software for manipulation and processing. Various strategies are then invoked to increase the actual communication rate.

Once an input sequence has been collated by the efficiency strategy, this is made available to the application software which uses this information. The application software may be specially designed for the user, e.g. a communication program, or it may be freely available, e.g. a wordprocessor, spreadsheet of games software.

The application software will govern the type of output generated, e.g. via a monitor or printer. Users with special needs may require alternative output forms. A partially sighted person may require large print on the monitor, whereas a blind user would need spoken output as well as braille print. A tactile output is also available.

As this dissertation is primarily concerned with input enhancement, the discussion on interfacing will be restricted to the following areas:

- physical input methods;
- selection techniques; and
- strategies for increasing the rate of communication.

INPUT METHODS

The nature of the disability will dictate what movement can be used to operate modified keyboards or specialized input devices in the form of switches (Heckathorne, 1986). Any reliable voluntary movement can be harnessed to operate an input device. Gross movements can activate resilient switches while very sensitive switches are used when movements are weak and specific. These switches are easily made, using keyboard switches and consumer items and applying electrical knowledge.

A variety of "off-the-shelf" switches on the market include touch-activated switches, tread switches, lever, leaf, sip/puff switches, light pens, joy-stick controls: some of which are remotely connected to the computer by radio or infra-red wireless links (Bates, 1985; Boonzaier & Kleviansky, 1987; French et al, 1987; Heckathorne, 1986). These switches are activated by any controlled movement which can be elicited in either upper-limb, lower-limb, trunk, head or facial movement. The mix of usable channels differs greatly from individual to individual.

Electronics are also used to detect slight muscle contractions by direct mechanical or myoelectric amplification techniques to activate a switch. Such an interface can reliably detect small 'twitches' of the eyebrow or any other controlled minor muscle movement. A method of detection where a user selects items visually, involves complex electronics to measure the angle of reflection on the cornea of the eye. Word/speech recognition is a powerful input mechanism which can be very reliable when limited vocabularies and consistent voice patterns are used. It

should be noted that voice control and not speech intelligibility is of importance. A speech defect need not influence the usefulness of this input method as long as the electronics demand only a consistent input.

The interpretation of the visual cortex electroencephalogram as an input source is currently under research (Sutter, 1983). By placing electrodes on the occipital region of the skull, it is possible to detect electrical wave patterns which correspond to the temporal visual information being processed at that time. At this stage spatial patterns cannot provide a reliable enough input and this research is still in its infancy.

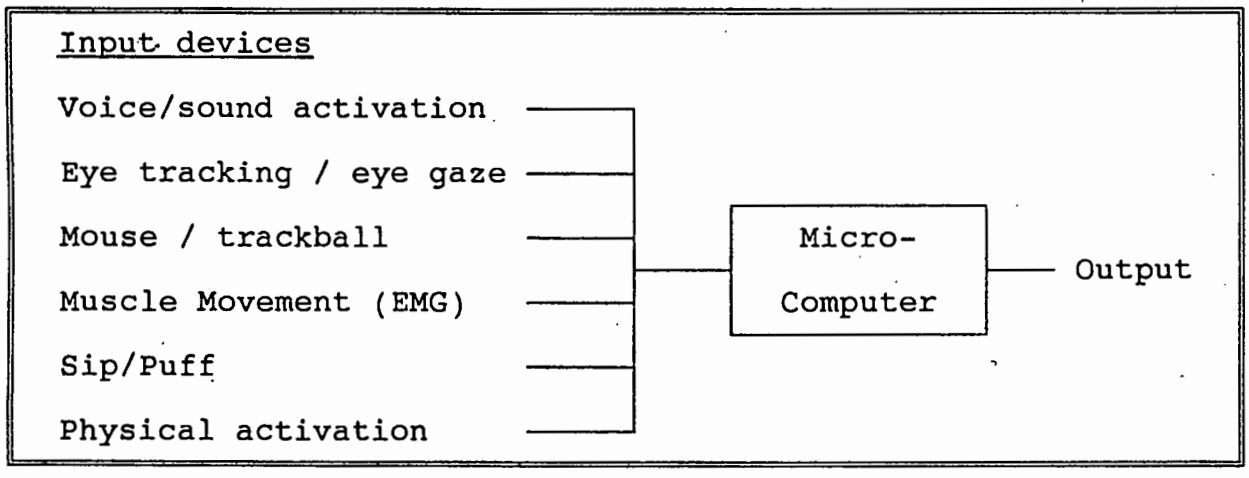


Fig: 3-2 Summary of Input Devices

INPUT SELECTION TECHNIQUES

Physically disabled individuals are often unable to access a keyboard in the conventional manner and alternative input selection techniques need to be employed (fig. 3-3).

When using a keyboard in the conventional manner, the operator is able to select any element of the symbol set, in this case letters, numbers and punctuation, by accessing the keys

'directly', sometimes two at a time. The direct selection technique (figure 3.3a) is the most efficient selection technique available. However, this method requires a high degree of dexterity in order to access a large symbol set. The **independent decision control (i.d.c.)**, i.e. the number of mutually exclusive choices which can be made, must equal, or exceed, the number of symbols (n) in the symbol set.

Direct selection is therefore only possible if:

$$i.d.c. \geq n \quad ...(1)$$

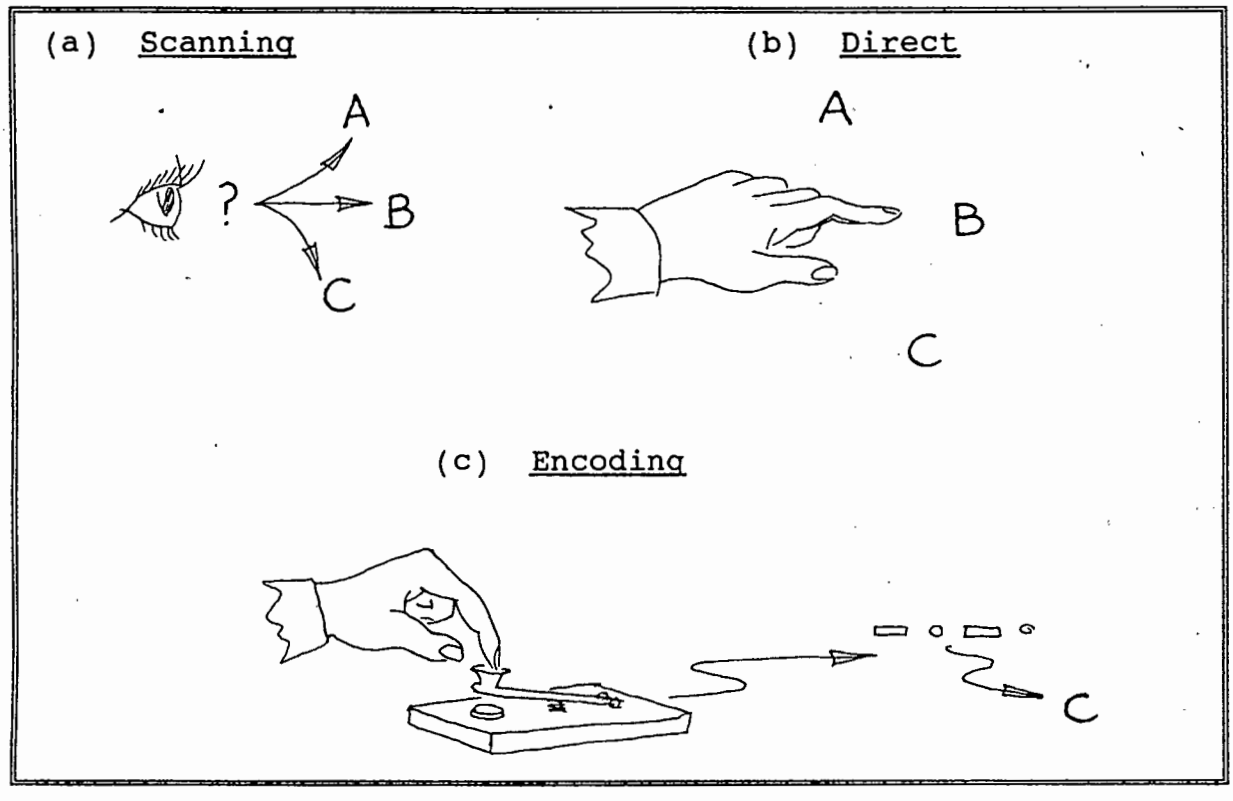


Fig: 3-3 Input Selection techniques

Speech is another example of direct selection as each symbol (word) can be accessed directly. Alternately, an augmentative communication board can be accessed using direct selection if the user can point directly to each symbol on the board. "Indication", i.e. pointing at an item, need not depend only on

an index finger. A fist, a toe, or eye-gaze techniques, can easily be used.

Problems arise when direct selection is not possible, i.e. when the number of symbols exceeds the independent decision control:

$$i.d.c. < n \quad ...(2)$$

The communicator now has to resort to a **scanning** selection technique (figure 3-3b). Scanning subdivides the symbol set into subsets which can be accessed by one degree of decision control, e.g. with a single switch.

Linear/circular scanning is used when only one independent decision control is available. The symbol set is subdivided into subsets of one element each. These are scanned linearly until the target symbol is selected. A conversation, analogous to the popular guessing game, Twenty Questions, in which the 'listener' has to elicit dialogue from the non-speaking partner by asking questions which can only be answered with either a "yes" or "no" response, is a scanning system. The selection technique in scanning requires very little movement and can be used in situations where physical abilities are very limited. Because scanning may necessitate long waiting periods before the required item is indicated, it is far slower than direct selection.

Multi-dimensional scanning allows for the selection of subsets of symbols, thereby increasing the scanning rate. Column-row, or row-column scanning organizes the symbol in matrix form. The

columns of the matrix are scanned serially, until the user selects the column in which the desired element appears. The individual items of the selected column are then scanned by row until the target symbol is selected. Most row-column scanners are two dimensional. A hierarchical system of subsets can be implemented by using certain symbols to indicate additional symbol subsets, which are themselves matrices.

Directed scanning is used if the communicator is able to control the movement of the cursor. This selection technique is often faster than automatic scanning as the user controls both the indication and selection of items. However, greater physical ability is needed as more than one switch may be used: a) to direct the cursor, and b) to select the symbol item.

Encoding (fig. 3-3c) is used in an attempt to speed up the selection technique. Encoding refers to methods which use a smaller coding symbol set to select symbols from a larger set of target symbols. Morse code is a well-known coding system which uses a two symbol sequence (dot, dash) to code the 27-symbol set of alphabetic characters (26 letters and the space). A transformation occurs between the coding symbol set and the target symbol set (i.e. the code is translated into a target symbol):

$$s_t = f(s_{c1} \ldots s_{ci})$$

where: s_t is a member of the target symbol set;

s_{cj} is the jth member of the coding symbol set;

f is the transformation function which maps coding symbols onto the target symbol set.

This transformation places a higher cognitive load on the user. The particular coding set used can comprise a mixture of any type of symbol, e.g. colours, numbers, pictures or letters, depending on the mental and physical capabilities of the user.

Until recently, encoding was viewed as a selection technique in itself (Vanderheiden, 1976). Confusion arose when attempting to define "scanning" and "encoding". It can be argued, for example, that Morse Code can be implemented using a scanning or direct selection technique. If the "scanning" technique is used, the 2-member dot-dash symbol set will be scanned alternately. The term, "direct selection" will be used if the dot and dash are selected by direct indication. However, Damper (1984) suggests that the criteria should be dependent on how critical the timing of inputs is. In scanning, the time is critical when selecting an element, whereas in an encoding technique the order of inputs is crucial. Vanderheiden and Lloyd (1986) have used encoding as a strategy for increasing overall the communication rate in either the scanning or direct selection techniques.

Encoding is more elaborate, but faster than the ordinary scanning procedure. It can be implemented in various forms: Memory-based encoding implies that the coding system is memorized

before use. Morse code, for example, can be memorized and used to encode letters. The communication rate is increased as the user does not need a look-up table. However, the cognitive requirements are also increased. The communication rate will decrease if the listener is not familiar with the system.

A **chart-based encoding** system reduces the need to memorize a coding system. A chart is referred to when a symbol is required. Both the user and the communicating partner can use the chart.

Display-based encoding systems refer to computer-based encoding systems. Various forms of encoding can be implemented as the display is able to change in response to different inputs. It is thus easier to implement a deep hierarchical encoding system on a computer-based device than on a manual system.

Encoding can take different forms and is the basis for techniques aimed at speeding up the communication rate. The importance of an increased communication rate has already been explained. The problem of increasing this rate, is at present the subject of much research, and forms the basis of this work.

STRATEGIES FOR INCREASING THE COMMUNICATION RATE

Various techniques are currently used to speed up the communication rate. These techniques will be introduced and specific implementations will be cited. Conversation and writing needs have different characteristics and the absence of either will result in communication problems (Vanderheiden, 1983). A distinction will be made between writing and conversation devices and the discussion will focus on these specific communication needs.

Orthographic spelling is an encoding technique familiar to the reader. Although this technique is open, i.e. there is no limit to what can be expressed, it tends to be the slowest of all communication systems, as each letter in every word has to be selected. English words require an average of 4.5 letters (Damper, 1984), thus requiring a minimum of five selections (four letters and a space) per word in the simplest linear scanning method.

Abbreviation techniques are used to expand mnemonic codes into words, sentences or phrases. These mnemonic codes are chosen so that they are identifiable with the expansions. Problems arise as the number of codes increase, making the recall process difficult. Most abbreviation techniques use orthographic letters which limit the eventual abbreviation base. Numeric codes can also be used, but it is unreasonable to expect a user to memorize a large vocabulary using such a system.

Semantic compaction is a technique devised by linguist Bruce Baker. Predominantly a conversational technique, semantic compaction allows a non-speaking person to retrieve a large vocabulary of sentences, by using an easily remembered meaning-based graphic coding system.

Anticipatory techniques are primitive and unwieldy at present. Statistical knowledge of language enables a selection of language units to be offered, thus increasing the communication rate. Savings of up to half this rate have been achieved but this often involves an increased cognitive load on the user.

Conversational Systems

Several methods of conversation exist. The fastest and most common is that of speech. When the spoken medium is incomprehensible, too slow, or not available, augmentative and alternative (non-speech) communication methods are used. Signs and gestures are widely used and communication also takes place through visual and printed forms. Although speech output is desirable as part of an alternative communication system because it is readily accepted by the listener, the actual communication rate is by far the more important factor in measuring effectiveness.

Use of classical sign language is not always possible for a physically disabled individual. Physical limitations may disguise the signs, making it difficult to recognize conventional signing. This, however, does not reduce the importance of signing as highly individualistic modified signs can be

recognized by those in close contact with the user. Uncontrolled and uncoordinated movement can complicate the recognition process and, unless the communicating partner is intimately acquainted with the augmentative communicator's techniques, confusion is likely to occur. Many severely disabled individuals have a useful vocabulary of signs which are often overlooked because of the overriding physical disability.

Because of the inefficiency of this type of signing, people resort to symbol boards. Symbol sets vary from simple pictures to orthographic letter boards (fig. 3-4). A number of graphic communication systems are available and reduce the professional's need to develop individual symbols. Basic picture systems include the Oakland Picture Dictionary and the Picture Communication System (PCS) (Johnson, 1985). The pictorial nature of these systems makes it difficult to portray emotional and abstract concepts. However, the concrete basis of these particular systems make them suited to the beginner as well as some mentally retarded individuals.

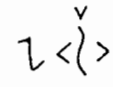
Picture Communication System	
Oakland Dictionary	
Blissymbolics	
Traditional Orthography	cold

Fig: 3-4 Symbol sets representing cold

Blissymbolics (MacNaughton, 1985; MacDonald, 1980) is a meaning-based graphic communication system originally developed as an international written language. Symbols are constructed

from a core vocabulary of pictographic, ideographic and arbitrary symbol components. The logical basis of this system allows for symbols expressing feelings and abstract concepts. A strict syntactic rule base results in a very powerful communication system which can be implemented at any level. Although Blissymbolics can mirror extremely complex language structures, it can also be used on a very simple communication level.

Various other graphic communication system are available. These include **Sigsymbols** - the written representation of signing (Loomis, et al, 1983); and **Worldsign** - an international form of signing (Orcutt, 1985). **Traditional orthography** is also used as a symbol set on letter and word communication boards.

Many communication systems have been implemented on microcomputers. Apart from easily duplicating the systems, resident software is able to to expand certain inputs in an attempt to increase the communication rate. Systems are also easily customized by using software parameters for individual users.

The severity of the disabilities experienced by non-speaking communicators often dictates a minimum of input switches. Computing power can be used to expand the minimum of input into a form of conversation which can be understood by the listener. Output usually involves a display on screen and printer, although speech output is a recommendation.

Semantic compaction, mentioned earlier, is a technique whereby conversational utterances are retrieved efficiently and simply.

The "MINSPEAK" system is an implementation of this technique and is available on either the "TOUCH TALKER" or the "LIGHT TALKER", developed by Prentke Romich. "MINSPEAK" provides a retrieval system by which pre-stored words, phrases or sentences can be recalled by pressing one or a combination of symbolic keys (Baker, 1982; Baker, 1984). "MINSPEAK" can utilize semantic information and can be personalized by the individual user. The principle of this system is to allow the same graphic symbols to be used in different ways. One method is to construct sentences which are retrieved by a specific, but logical combination of elements. The pictorial symbols serve as a rich aide-memoire to the stored sentence. As an example, consider the symbols in figure 3-5.

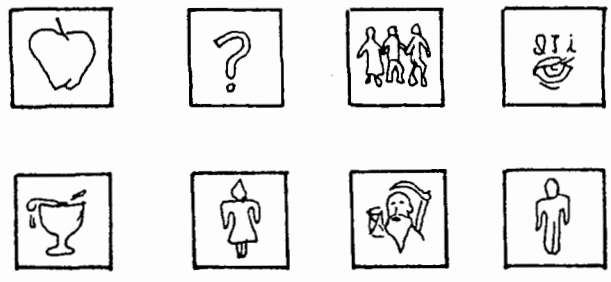


Fig: 3-5 A selection of MINSPEAK keys.

To say " I see them", the disabled user would select the symbol combination in figure 3-6. This encoding is obvious to the human mind, but not to the computer. The computer is therefore not being used to interpret language, but rather to reference a pre-defined combination of keys to a corresponding, previously stored sentence, and to output this as speech.

Fig: 3-6 "I see them."

If the user is able to read, the symbols can have letters and numbers to add richer meaning and to lessen memorization. For example, the letter in the symbol of figure 3-7 is associated with "algebra".

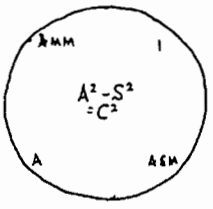


Fig: 3-7 A symbol easily associated with "algebra".

In the event of the system becoming too unwieldly, the concept of themes (called "MINTHEMES") can be introduced. If, for example, the apple symbol (associated with food) is selected twice, "food" becomes the "MINTHEME" context. The sequence in figure 3-8 would denote "you're feeding me too quickly".

Fig: 3-8 "You're feeding me too quickly."

Retaining this symbol theme to provide the semantic context, the single selection of the symbol in figure 3-9 would cause the pre-stored sentence, "Food is getting on my clothes", to be spoken.

Fig: 3-9 "Food is getting on my clothes."

"Word Strategies" is a further development of "MINSPEAK". Instead of storing complete sentences, this software enables the user to construct new sentences by modifying open language phrases. Verb and person modification and other linguistic manipulations can be performed with minimal input. Savings over conventional orthographic selection are approximately 60% (Baker, 1987).

Writing Devices

Informal writing constitutes the major portion of written work. This is obvious when one thinks in terms of any occupational output, e.g. a school child doing class- and homework, and a businessman compiling reports. The efficient creation of written work is even more important to a disabled individual, who may not be able to use his hands, therefore excluding occupations which require manual tasks.

Although writing needs do not require the speed necessary for conversation. Ordinary typing or, in most cases, wordprocessing speeds by a disabled typist of 5 to 10 wpm, is still way below the acceptable rate of 30 to 35 wpm. However, this can be speeded up twice and even three times when using the computer's processing powers (Vanderheiden, 1983)

For the minimally physically disabled individual, a typewriter is often sufficient for writing purposes. Poor hand co-ordination and erratic movements result in poor typing accuracy. The availability of correcting fluids and tape, and editing

screens, helps to solve the accuracy problem. However, the increased stress and anxiety of having to feed paper into a typewriter, and correcting typographical errors manually, reduces the actual typing speed and therefore the efficiency still further, and often leads to avoidance of the task.

With the advent of the microcomputer, affordable wordprocessing is now available to many disabled individuals. Much of the physical stress is reduced because of automatic correction and the ability to store text until it is printed. Commercially available wordprocessors such as "WORD-WISE" (trademark BBC), "WRITING-ASSISTANT" (trademark IBM) and "APPLE WRITER" (trademark APPLE) are being used by disabled individuals.

A commercially available software package for the IBM-PC allows the user to re-configure the keyboard (Vanderheiden, 1982, 1984; Jaros, et al, 1987). This feature encourages the storing of commonly-used words and phrases under one or more keystrokes, thereby increasing the typing rate. Another feature modifies the keyboard interrupt handler so that it accepts any "control", "alt" or "shift" sequence sequentially instead of simultaneously. A typist using only one finger, or a mouthstick/headpointer, can then issue a dual keystroke as two sequential keystrokes. This capability allows "one-finger" typists to operate a wide variety of applications software.

Many severely disabled individuals either type very slowly or are incapable of using a conventional keyboard. For this category of typist, special wordprocessors have been developed. Most of this software implements multi-dimensional scanning by

presenting matrices of orthographic characters. These wordprocessors also use special functions to increase the obviously slow communication rate.

The most common techniques used to increase the communication rate rely on the inherent statistical occurrences of letters in a given language. The multi-dimensional letter matrix is structured so that the most-used letters are placed close to the start of the scanning procedure (fig. 3-10).

space	E	return	L	S	B
T	A	I	M	H	K
O	N	D	F	G	V
R	C	Y	X	J	Ω
U	P	Q	Z	W	

Fig: 3-10 A static letter matrix

Variations of the static matrix are used in practice. For instance, the matrix in figure 3-11 is used by a non-speaking person for conversation (Clarke, 1987). The matrix has been refined over a long period and reflects the user's individual language usage. Instead of using an electronic device to select letters, the user indicates row-column selection with his eyes.

space	I	A	O	R	S	T	H
E	M	N	U	G	B	V	G
return	Y	D	P	C	F	W	Z
		L	K	J	X	Q	

Fig: 3-11 Conversation letter matrix

"MAC-APPLE" is a specialized wordprocessor which is suitable for use by disabled people (Poon, 1985). This system allows the user to choose between scanning, encoding or direct selection techniques. Any individual, no matter how physically disabled, can thus use the system.

The "MAC-APPLE" system caters for all text processing needs, but offers special functions which attempt to increase the typing rate. One such function inserts two blank characters into the text, and sets the shift key to uppercase for the next character, whenever a fullstop (period) is selected. This function also inserts a blank after a comma and inserts the character "u" after a "q". "MAC-APPLE" offers an arithmetic mode whereby a student can arrange numbers in the middle of the screen. The cursor begins at the units column, allowing the user to manipulate his numbers in a logical right-to-left sequence. A very useful function displays word-lists for selection without having to type entire words. These word-lists are presented in either an alphabetical sequence, if the first letters are already selected, or in the order in which they were first entered (an "unintelligent" method).

It is statistically possible to predict possible subsequent letters depending on the probabilities relating to the preceding input. This knowledge allows short lists of probable letters to be displayed in the order of probability. The user then makes his choice of letter, by means of his particular selection method. This theory has been used to devise the Communication and Device Control (CDC) system (Heckathorne & Childress, 1983). The CDC offers letter and word selection, editing and

environmental control.

In letter selection (figure 3-12), five letters are displayed on the screen. These letters are chosen according to the probability that they occur after the preceding letters. The sequence of options is calculated from statistical probabilities. If the required letter is not presented as a choice, the user asks for the next five letters by selecting the "*" symbol. The three end characters, ",", "#" and "@", are branch points to other system operating modes.

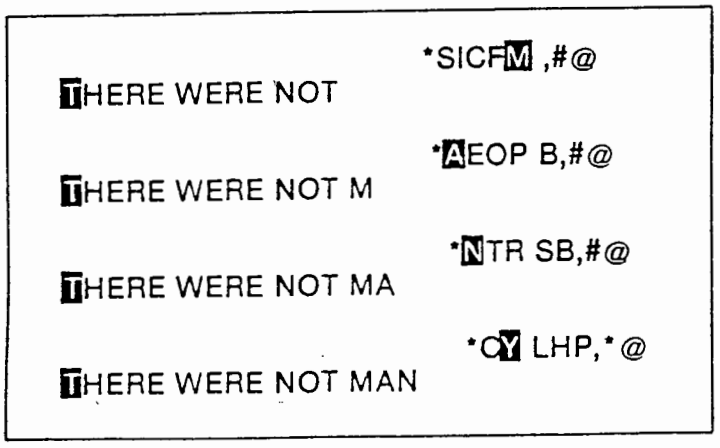


Fig: 3-12 An example of letter anticipation.
(Note how the space is also anticipated).

Word selection operates in a similar fashion, except that it utilizes a brute-force method rather than an anticipatory one. However, this procedure has the ability to "learn" new words by adding them to the dictionary. The letter and word selection procedures thus complement each other (fig. 3-13).

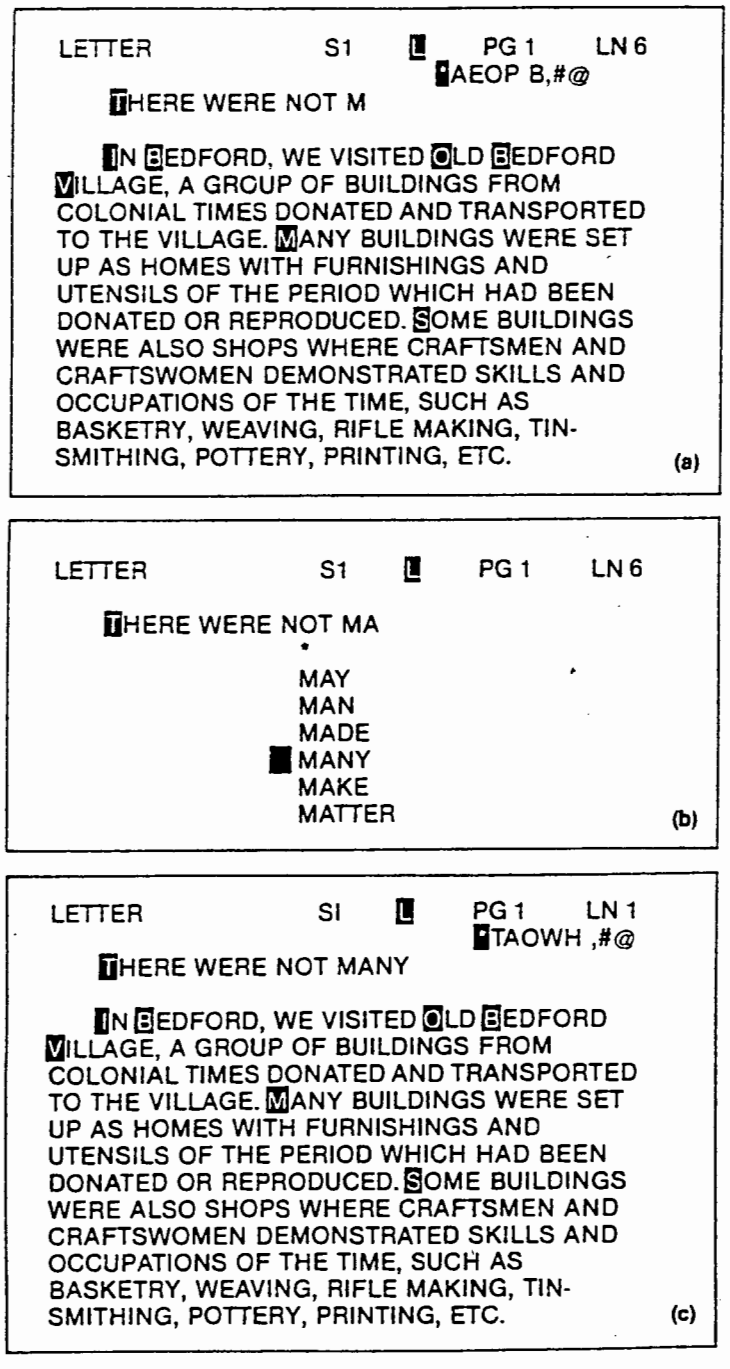


Fig: 3-13 A text-generation sequence with word anticipation. The user intends to generate the word "MANY" and begins by selecting the first two letters, "MA" (a). The page is replaced by a list of words from the dictionary (b). The user then selects the word, which is automatically completed in the buffer line (c). (The flashing cursor is seen scanning the lists. All other inverse characters denote capital letters.)

Another anticipatory communication aid is the Predictive Adaptive Lexicon based interface - PAL (Pickering et al, 1984). This device anticipates the next word from previously typed characters and historical word usage. The anticipated word is rejected if further letters are chosen. The prototype system consisted of a rudimentary word processor. Despite a concern that additional cognitive load might be a detrimental factor, savings in excess of 30% were reported. In other anticipatory systems, savings of up to 50% have been reported (Witten et al, 1982).

It is felt that this technique, known as predictive or anticipatory text selection, offers the best potential for a communication program, to enable disabled individuals to produce written work at an acceptable rate.

CHAPTER FOUR

EFFICIENT INPUT SYSTEMS

INTRODUCTION

The previous chapter introduced the concept of increasing the communication rate by analyzing letter and word frequencies. Further investigation into functional linguistic theory highlights additional language structures which can supplement and improve current anticipatory communication techniques.

A detailed explanation of these techniques follows in which the difference between letter, word and phrase anticipation is discussed. This in turn is followed by a discussion on the relevance of linguistic structure of language to anticipatory text processing, and an introduction to the analysis of the communication rate:

- input techniques;
- linguistic relevance to text analysis;
- communication rate.

INPUT TECHNIQUES

Various attempts have been made to provide efficient input systems for disabled individuals. The factor which most affects the efficiency of a communication system is the rate at which communication takes place. Attempts used to increase this rate are divided into two strategy types: expansion and anticipation (also called prediction).

Expansion techniques are especially suited to microcomputer-based implementations. The speed of such a system can expand a code within milliseconds thus facilitating a more acceptable communication rate. Instead of selecting every component of a message, the user is able to produce the same message by selecting a minimum of components. The computer software then expands the selection into the desired message. For example, "e", "g" might be expanded into the phrase, "for example".

Encoding systems are examples of symbol expansion. Communication systems using abbreviation expansion rely on letter or numeric codes to denote words and phrases (Damper et al, 1986). Semantic compaction (Baker, 1985, 1987) uses a different approach where unique strings of user-defined graphic symbols are chosen to represent words, phrases and sentences.

Stenographic language uses phonetic codes as symbols (Arnott 1987, Steele, 1983) which can later be expanded into orthography. However, when a stenograph is used to produce phonetic speech, it cannot be referred to as expansion system as each code represents a specific sound and does not 'expand' the target symbol, but merely translates it into a language unit, often of similar complexity. Likewise, Morse Code translates a complex combination of symbols into a single orthographic letter.

Expansion techniques tend to be 'brute-force' as they are basically "look-up" methods. Communication rates are increased tenfold as a minimum of selections are required for very long messages. Although mnemonic codes are used in orthographic abbreviation systems and graphic codes are themselves mnemonic,

the number of codes that can be remembered are limited.

Anticipatory systems rely more on a 'knowledge' of the communication system's linguistic framework. Instead of just translating a coding system into an expanded form, the preceding symbol selections are scrutinized and the consequent symbols are anticipated. This technique has been observed in manual scanning systems where the 'listener' is able to suggest (anticipate) the next symbol intuitively. Not only does this restrict the symbol set, it also increases the communication rate.

Anticipatory techniques are not as "brute-force" as expansion techniques, and can be divided into purely statistical methods and methods relying on linguistic knowledge. They also attempt to incorporate the native speaker's intuitive knowledge of the symbol system.

Most anticipatory communication systems make use of global language statistics and cater for both letter and word anticipation (Bentrup, 1987; Beletsa, 1977; DeRoost, 1986; Gibler 1981).

Artificial intelligence techniques are being used to harness the redundancy in English in attempt to increase the communication rate of disabled typists. One such implementation proposes that the number of keys used to represent the alphabet be reduced from 27 to 11 (Foulds et al, 1987; Levine et al, 1987). The proposed keyboard employs nine keys which each represent a cluster of three letters, a tenth key dedicated to the space, and an eleventh used for error correction. The model

uses fourth-order transitional probabilities to disambiguate among the characters of the three level clusters.

In the ensuing discussion, reference will be made to scanning systems. This is for convenience only, as all discussions are also applicable to systems employing direct selection.

LETTER AND WORD ANTICIPATION

The most common use of anticipation is in the basic letter matrix which is in use in many scanning wordprocessors. Although the matrix is static, i.e. the individual letters do not change their position, the letters are arranged according to global letter frequency. Thus, more frequently used letters are situated near to the scanning start of the matrix, minimizing scanning time.

Word anticipation has increased communication rates considerably and usually takes the form of dictionaries (Gibler, 1981; Swiffen et al, 1987; Treviranus & Norris, 1987). The implementation of dictionaries range from static word lists, which can be updated by the user, to statistical analyses of most commonly used words. The term 'word' can also represent morphemes and phrases. "MacApple" [1] allows for the coupling of prefixes and suffixes to any word currently being processed.

1. See page 49 for description of "MacApple".

IMPLICATIONS OF SEARCH TIME

Dynamic letter matrices differ from their static counterparts in that the matrices change after each letter selection. Because the display changes, an additional cognitive load is demanded in terms of searching (Soede & Foulds, 1986; Treviranus & Norris, 1987).

In a static display, the user knows where the target symbol is situated. Before the scanning mechanism begins, the user has already focussed on the target and can thus wait for the desired element to be indicated.

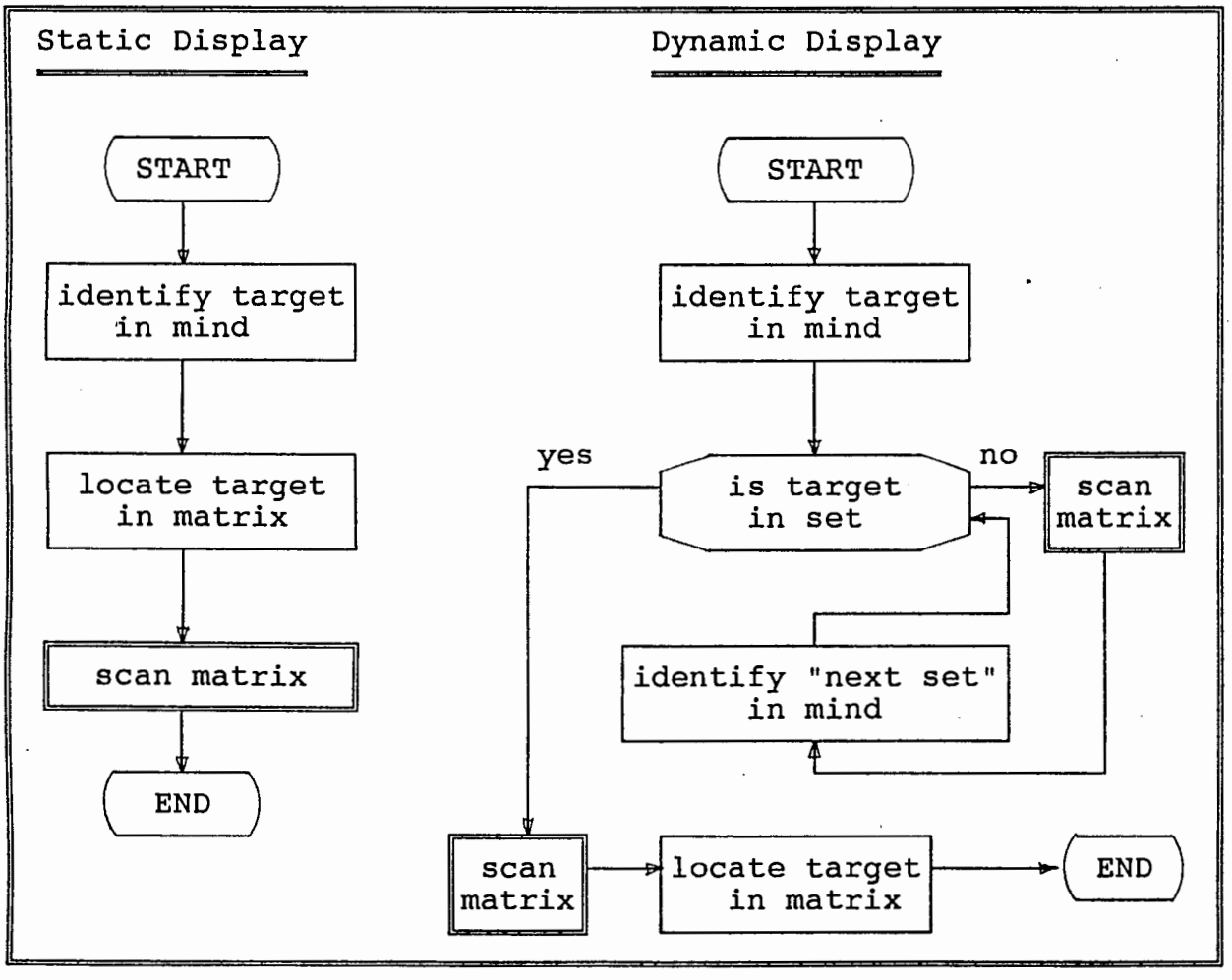


Fig: 4-1 Flowcharts representing Static and Dynamic Displays

When using a dynamic matrix, the user has to locate the target symbol first. In an attempt to lower the cognitive load, anticipatory systems do not display the entire symbol set at once, but offer the user a subset of the most probable symbols. This simplifies the selection task as the user has to locate the target symbol within a smaller set. If the target symbol is not there, an additional set is produced. Users still have a cognitive overhead in that they have to search for the location of a symbol before scanning begins. A delay at the beginning of the scanning process is necessary to provide the necessary search time. Some researchers feel that the scanning mechanism should speed up as it moves through the symbol set, as the further away the target is, the more time is available for searching (Heckathorne, 1985). Although this is valid in theory, the physical limitations of the user often dictate the optimum speed of the scanner. Reaction time may prevent varying speeds within a scanning activity.

LINGUISTIC RELEVANCE TO TEXT ANALYSIS

Spelling in natural languages such as English shows redundancy. This often helps to ensure that no ambiguity results. For example, the words *sea* and *see* are pronounced the same way, but have different meanings. Modifications often occur when letters do not affect meaning and are purely redundant. American English is noted for its "streamlining" of the English language: *Programme* has become *program*, *colour* has become *color*, and *catalogue* has become

catalog. [1]

Zipf's Law of Least Effort is fundamental to increasing the communication rates of physically disabled individuals. The physical effort required by the information sender must be minimized, while retaining the content of the message. In conversational situations it is sufficient to convey the meaning of a message. However, in orthographical situations, it is necessary to adhere to syntactic rules governing spelling and word order. It is thus essential that the typist be able to minimize the physical activity needed for formal writing.

Linguists and psychologists have shown an interest in text analyses for decades. Markov (1913) proposed that language can be modelled as a stationary stochastic process, where successive symbols are selected from a finite vocabulary based on fixed probabilities. Finite state automata are used to illustrate the process of moving from one state to another, depending on probabilities. This very simplified view is used in the so-called n-gram predictions, as the occurrence of letters depend on the preceding letters.

This dependence was analysed statistically by Shannon (1951) who used probability occurrences to predict the entropy distribution and redundancy of English. The redundancy in English spelling is high, approaching 60%. This redundancy can

1. Is this not an instance of Zipf's Law of Least Effort? The underlying goal of which is to communicate an idea effectively using as little effort as possible by the sender. Zipf postulated that language speakers strive to reduce the amount of words, and letters per word, needed to convey a message.

be seen in the following text which was generated by a disabled typist from dictation over the telephone. The object was to write a number of captions, which were to appear below a series of photographs depicting disabled children using various communication devices.

> at last i can mak mys undsdt . wo my c i wd nmot even be ablr to greet you, but now i can enjoy a c0onv w my friends
> iui cnnt vtalk , use my hads or red , but can say whatev i want to my frds using pics 9blissymbol0 oon a brd. [2]

Despite the high error rate [3], it is still possible to understand the gist of the passage. Three types of errors can be identified: Minimal punctuation; abbreviations; and mistyping. Note the different techniques used for abbreviations:

- 9 and 0 in place of left and right parentheses located at shift 9 and shift 0 respectively;

- partial absence of vowels;

- absence of capital letters.

The ability to translate the meaning is simplified if the reader can identify with the context of the conversation. Having

2. Transcription of actual typing produced by a disabled person receiving dictation during a telephonic conversation (personal experience).

3. The use of the term "error rate" in this context does not refer solely to classical errors, as the transcript includes deliberate shortcuts.

completed the telephone conversation, the disabled typist was able to edit the text as follows:

> At last I can make myself understood. Without my computer I would not even be able to greet you, but now I can enjoy a conversation with my friends. I cannot talk, use my hands or read, but can say whatever I want to my friends using pictures (Blissymbols) on a board.

There is also a significant difference between the redundancy factor of vowels and consonants (Fry, 1960). Consider the following sentences:

D- y-- c-m- h-r- -ft-n ? ...(1)
I -i- -o- --i-- -o. ...(2)

In sentence (1), all vowels were omitted, whereas the consonants were omitted in the second example. It is a simple exercise to decipher (1) as being "Do you come here often?". Sentence (2) is however almost impossible to decipher as "I did not think so.". The first sentence contained a larger symbol set, whereas only five symbols were used to code message (2). The more information supplied, the easier a message is likely to be understood (Newman, 1951). It is also apparent that the more redundant the system is, the easier the translation. In (1), very little possibility for misunderstanding occurred. However, the message sender has to send a longer message, thereby reducing the communication rate.

Once again, communication rate poses a problem. How can this rate be increased? Can the redundancy factor be employed?

STATISTICAL ANALYSIS OF LETTER FREQUENCIES

Words are not an arbitrary sequence of possible letters. This has been shown by statistical analysis of letter frequencies (Bourne & Ford, 1961; Denes, 1963; Fang, 1966; Fraprie, 1950; Gerber & Vertin, 1969; Solso, 1979; Wang & Crawford, 1960). Further investigations have identified different ways in which letter frequencies can be analyzed. These include the identification of: phoneme (sound unit) clusters (Carroll, 1938; Fowler, 1957; Gibson et al, 1962; Siguard, 1968); word-initial and word-terminal letters (Solso, et al., 1982); and consonant clusters (Keller & Saporta, 1957).

CbATSIPOCMRFWBDELHVNGUJKQYZXb	CNETUAbBCDFGHIJKLMNOPQRSVWXYZ
CALNTURSMPDbBVCIKEYGFJHOZAQWX	CONMURLVSGPCTAbDOWHBIFEXQKYJZ
CBIESbABCDFGHJKLMNOPQRTUVWXYZ	CPIbHABCDEFGJKLMNOPQRSTUVWXYZ
CCEDUAILbRHBCDFGJKMNPQSTVWXYZ	CQUbABCDEFGHIJKLMNOPQRSTVWXYZ
CDObABCDEFGHIJKLMNPQRSTUVWXYZ	CRRIOAUYbBCDFGHJKLMNPQRSTUWXZ
CEbSNRDPLIEMAFTCBYOKUZGHJQVWX	CSbTOSABCDEFGHIJKLMNPQRUVWXYZ
CFbAEORBCDFGHIJKLMNPQSTUVWXYZ	CTIbESOURLAFNMVTBCDGHJKPQWXYZ
CGObABCDEFGHIJKLMNPQRSTUVWXYZ	CULRSTMPBEIAONUDFCJbGHKQVWXYZ
CHbAIOENRULMTYSWFBCDGHKJPQVXZ	CVEbABCDFGHIJKLMNOPQRSTUVWXYZ
CIAEPOSTNDLRFVZBUMGbCHIYJKQWX	CWbABCDEFGHIJKLMNOPQRSTUVWXYZ
CJAbBCDEFGHIJKLMNOPQRSTUVWXYZ	CXXbABCDEFGHIJKLMNOPQRSTUVWYZ
CKbEISLYNGABOHWTPFMRUCDJKQVXZ	CYbTCANPLMSBDREFGHIJKOQUVWXYZ
CLEAUOIYbBCDFGHJKLMNPQRSTVWXZ	CZEbABCDFGHIJKLMNOPQRSTUVWXYZ
CMbEABCDFGHIJKLMNOPQRSTUVWXYZ	

Fig: 4-2 Digram listing for digrams beginning with /C/, where b represents the space character. (After: Gibler, 1981)

Tables which identify the frequency of letters in relation to the preceding letters have been compiled (fig. 4-2). These statistics are referred to as n-gram probabilities as they list the probability of each letter in the alphabet to follow a sequence of n (number of) letters. It is then possible to predict the most likely letter to occur after a sequence of n

letters (Solso et al, 1982). The longer the sequence, the more accurate the anticipation will be (Newman, 1951). N-gram tables have been compiled for single letter and digram frequencies (Mayzner & Tresselt, 1965; Rawlinson, 1976), and trigram, tetragram and pentagram frequencies (Mayzner et al, 1965). Every combination of n letters is listed, followed by a series of letters in the probability order in which they will occur. Word-initial and word-final n-gram frequencies have also been calculated (Rubin, 1978).

Some anticipatory mechanisms rely on n-gram predictions (Gibler, 1982, 1983; Childress, et al, 1982). For every sequence of n letters, the probability of the following letter is calculated. Figure 4-2 illustrates the probability order in which letters occur after digrams beginning with /C/. The first five predictions to be offered after the digram /CH/ are <space>, /A/, /I/, /O/, and /E/. These n-gram tables are computed from a selection of texts, and are thus relatively static. No account is taken of articulation conventions which could further refine the predictions. Gibler (1981) states that although quadgrams are the most efficient n-gram level, the difference in accuracy between trigram and quadgram is not significant enough to counteract the increased memory requirements and computation time. [4]

4. The physical limitations of present computer hardware still dictate the methods of implementation, to a large extent.

COMMUNICATION RATE

The quantification of communication rate is needed for the objective analysis of communication systems. Two measurements described by Gibler (1981) are communication fluency and efficiency.

Fluency is a measure of the range of output of which the communication system is capable. It thus measures the degree of openness of a system. The smaller the language elements, the more fluent the communication aid is. A communication aid, which offers the user all the letters comprising the alphabet, has the highest degree of fluency. This is because the user is able to construct any language element using the aid. However, an aid which presents the user with a limited set of language elements is very restrictive and thus has a low degree of fluency. The aim of any aid should be to have as high a degree of fluency as possible. However, there is one overriding disadvantage. A very fluent communication aid implies the use of small language elements. This, in turn, implies a very slow communication rate.

The efficiency of a communication system is measured by the average number of switch activations required to select a language element. The communication rate is measured in terms of the time needed for each language element selection. The time required to activate a switch is primarily dependent on the physical abilities of the user. Additional factors include the cognitive decision-making process needed to initiate the physical activation. These factors are, however, very difficult to quantify and are included in the time required for the

activation of a switch.

The more efficient a system is, the easier it is for the user to achieve a good communication rate. This is because the target items are more accessible. However, an efficient communication system usually requires a degree of anticipation which results in dynamic displays, which in turn results in an increase in search time and cognitive load.

The physical abilities of the user determine the number of input switches which can be used. A large number of switches increases the communication rate. However, this increase in physical activity and concentration places increased demands on the disabled user, which causes physical fatigue. This fatigue can decrease the communication rate as the physical ability to activate switches, is diminished. An attempt to increase communication rate is therefore dependent on balancing the efficiency of the communication system with ergonomic considerations.

MEASUREMENT OF COMMUNICATION RATE

The efficiency of a communication system is calculated as the number of scans needed to choose an element (Damper, 1984).

In order to measure efficiency quantitatively, a cost function is used to indicate the number of switch activations needed to select a specific target. For instance, the global statistical layout (Poon, 1983) and its associated cost matrix are depicted in figure 4-3. It takes four scans to locate "A", whereas only two are needed for "E".

space	E	return	L	S	B
T	A	I	M	H	K
O	N	D	F	G	V
R	C	Y	X	J	
U	P	Q	Z	W	

2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	
6	7	8	9	10	

Fig: 4.3 Static Statistical Layout with Cost Matrix

The conversational matrix discussed in the previous chapter has different costs associated with each letter (Fig 4-4).

space	I	A	O	R	S	T	H
E	M	N	U	G	B	V	G
return	Y	D	P	C	F	W	Z
	L	K	J	X	Q		

2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
	6	7	8	9	10		

Fig: 4.4 Conversational Layout with Cost Matrix

The object is to position the most likely elements in the positions of least cost. The efficiency is then increased because less input activations are needed.

Using the global frequencies of English letters, it is possible to compute the overall efficiency for both systems (fig. 4-5). The communication efficiency, J, is defined as the first moment of cost and letter probability:

$$J = \sum_{i=1}^{N} P(i)\, w(i)$$

where N = the number of letters in the group;
P(i) = the probability of the ith letter;
w(i) = the cost of the ith letter.

Letter	P(i) x 10^3	Static w(i)	Statistical P(i)w(i)	Conversational w(i)	Conversational P(i)w(i)
space	174	2	348	2	348
E	108	3	324	3	324
T	87	3	261	8	696
A	67	4	268	4	268
O	66	4	264	5	330
N	59	5	295	5	295
R	56	5	280	6	336
I	52	5	260	3	156
S	50	6	300	7	350
H	43	7	301	9	387
D	31	6	186	6	186
L	28	5	140	6	168
F	24	7	168	9	216
C	23	6	138	7	161
M	21	6	126	4	84
U	20	6	120	6	120
G	16	8	128	7	112
Y	16	7	112	5	80
P	16	7	112	7	112
W	13	10	130	10	130
B	12	7	84	8	96
V	8	9	72	9	72
K	2	8	16	7	14
X	1	9	9	9	9
J	1	9	9	8	8
Q	1	8	8	10	10
Z	1	9	9	11	11
$\sum_{i=1}^{N} P(i)\, w(i) =$			4468		5079

Table: 4-5 Letter probabilities and the number of cursor-steps required to select a character in the static statistical and conversational displays.

Thus, from table 4-5,

$J = 4.47$ for the static statistical system

and

$J = 5.08$ for the conversational system.

The static statistical system is thus theoretically more efficient than the conversational system.

When probability statistics are used, the cumulative probability of the most likely letters to follow a (n-1)-gram is used (Gibler, 1981). The cumulative probability of the R most likely letters to follow in a trigram is defined as:

$$Pcum\ (R) = \sum_{i=1}^{27} \sum_{j=1}^{27} \sum_{k=1}^{R} Pij(k)$$

where Pij(k) is the probability of letter k following the digram ij.

Similarly, the cumulative probabilities of digrams and monograms are as follows:

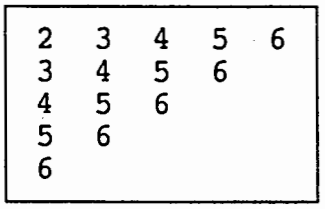
$$P_{cum}\ (R) = \sum_{i=1}^{27} \sum_{j=1}^{R} Pi(j) \quad \text{for digrams and}$$

$$P_{cum}\ (R) = \sum_{i=1}^{R} P(i) \quad \text{for monograms}$$

In Gibler's (1981) cost analysis, a two switch activation - directed scanning - was used. In this case, the cost matrix is as follows:

2	3	4	5	6
3	4	5	6	
4	5	6		
5	6			
6				

For a one-input system, an automatic scanner is used. The same basis for cost determination is used. However, instead of physically advancing the scanning device, the user passively waits for the scan to reach the target item. There is therefore less physical fatigue involved, but the cognitive waiting time is increased. The user is however given a greater search time. Instead of measuring switch activations, the number of unused scans should be analysed as the number of switch activations needed to select an item is always constant, e.g. 2 in the case of a row-column scanner.



Using the definition for J, the efficiency for monograms, digrams and trigrams as reported by Gibler (1981) are follows:

$$J = \sum_{i=1}^{27} P(i)\, w(i)$$

where P(i) = the probability of the ith letter;
w(i) = the cost of the ith letter.

J = 4.3 for monograms
= 3.6 for digrams
= 3.21 for trigrams

Although there is an increase in communication efficiency, the memory needed to store larger tables increases logarithmically with n For monograms, 27 bytes are required; for digrams, 279; and for trigrams 19,683. However, further analysis indicated that the most probable 3 letters in a trigram set constitute 66% of letter usage. Trigram implementations therefore store only

these first three letters, and are supplemented by digram and monogram tables (Gibler, 1981).

CHAPTER FIVE

LINGUISTIC CONSIDERATIONS

INTRODUCTION

The purpose of linguistics - the scientific study of language - is to describe and explain the structure and use of language (Falk, 1978). Computer science researchers rely heavily on linguistics when working in fields such as electronic speech synthesis, recognition and natural language parsing. The major use of linguistics, however, is in the field of artificial intelligence which strives for a better understanding of how the human mind operates (Schank, 1985; Minsky, 1985; McKean, 1985).

Natural language is a mental phenomenon, a body of knowledge about sounds, meanings and syntax which resides in the mind (Falk, 1978). Chomsky holds that the use of language stems from an innate knowledge of the syntax (structure) and semantics (meaning) of language, i.e. there exists a language knowledge (rule) base which resides in the mind (Lyons, 1970). An understanding of language is therefore helpful in the study of thought processes (psycholinguistics).

Linguistics is of particular interest when investigating language anticipation - the ability to anticipate the next linguistic item. Research into articulation mechanisms and phonology (the study of sound combinations) has revealed a series of rules which govern the formation of English words (Carroll, 1958; Denes, 1963). This theory, and its relevance to writing, is discussed in order to supplement the existing theories used in anticipatory wordprocessors for the disabled.

WRITING

Writing can be based on the sound system of a language - i.e. a **phonological** system - or it can be semantically based - i.e. a **morphological** system (Falk, 1978). An orthographical system, e.g. English, comes closest to the natural units of speech because the written symbols tend to correspond to the phonological system of the language (Newman & Gertsmann, 1952). Some languages, e.g. Cantonese and Mandarin, use a morphological writing system where the written units correspond to meaning rather than sound. **Syllabic writing systems**, e.g. Japanese, also exist in which each symbol represents a syllable - a phonological unit in which sound segments are combined (Falk, 1978). In the ensuing discussion, English will be used unless otherwise indicated.

There is often a discrepancy between pronunciation and orthography. For example, although the cluster /ps/ is not natural to English, it is nevertheless reflected in a borrowed word, e.g. psalm while the English pronunciation is [sa:m].

The phonological system of every language is dynamic and changes constantly (Falk, 1978). The written representation of language is however far more static. For instance, consider the words, lightand night The /gh/ phoneme is no longer in use as the pronunciation has changed, but the spelling still reflects the use of the unvoiced velar fricative, /x/ which existed in Old English.

Despite these anomalies, English speakers are able to recognize the phonemic representation of every morpheme in their writing system (Falk, 1978). Although English spelling may seem hazardous and inconsistent, phonetic rules do have equivalents in orthography (Dolby & Resnikoff, 1964).

THE STRUCTURE OF WORDS

A word is defined as any unit of language that, in writing, appears between spaces, or between a space and a hyphen. (Falk, 1978). Although Falk admits that this definition is not exact - e.g. the spelling of matchbox, match-boxand match boxare all considered to be correct - it will suffice for this discussion.

A native English speaker [1], attempting to read the list of words in table 5-1 below, would know that btcink is not phonetically permissible in English. However the speaker may have to resort to a dictionary in order to verify the existence of lickop

btcink	lickop	stin
srtokp	mahjui	ughytgr
lillon	vegtj	hunny

Table: 5-1 Fictional English words.

1. Knopf suggests that speakers are able to analyze phonetic rules from the age of three (1981).

Three types of sound sequences can be identified within English, as well as many other languages (Hawkins, 1984):

1. Sequences which actually occur in a language (e.g. *still*, *bring* in English);
2. Sequences which could occur, but do not exist yet (e.g. *stin*, *fleg* in English);
3. Sequences which do not occur because they violate the phonological patterns of the language (e.g. *bcint*, *srtokp* in English).

A PHONOLOGICAL BASIS

The sound segments which occur in English can be divided into four main classes: vowels, consonants, liquids and glides. The liquids and glides form a class referred to as approximants (Hawkins, 1984). The different classes are identified in table 5-2 below:

Name of class	Sound segments
vowels	[i, ɪ, e, ɛ, æ, ə, u, ʊ, o, ɔ, a]
consonants	[p, b, t, d, k, g, m, n, ŋ, f, v, θ, ð, s, z, š, ž, č, ǰ]
liquids	[l, r]
glides (semivowels)	[h, w, y]

Table: 5-2 Classes of sound segments

Table 5-3 lists the notation used to represent phonetic sound segments in English:

VOWELS			
Phonetic Symbol	Found in:	Phonetic Symbol	Found in:
[i]	beat, see	[u]	boot, suit
[I]	bit, sit	[ʊ]	book, soot
[e]	bait, say	[o]	coat, soak
[ɛ]	bet, set	[ɔ]	caught, sought
[æ]	bat, sat	[a]	cot, soft
[ə]	but, up		

DIPHTHONGS [2]		
[ay]	cry,	eye
[aw]	out,	cow
[ɔy]	boys,	coin

CONSONANTS		
Phonetic Symbol found in:	First Sound	Last Sound
[p]	peg	tap
[b]	bat	tab
[t]	time	hat
[d]	dog	rod
[k]	cat	knock
[g]	good	bug
[m]	might	lamb
[n]	no	seen
[ŋ]	—	thing
[f]	fetch	left
[v]	vine	of
[θ]	thick	path
[ð]	that	bathe
[s]	see	race
[š]	shoe	bush
[ž]	—	rouge
[č]	chin	patch
[ǰ]	just	ridge
[l]	lead	fill
[r]	read	fear
[h]	hard	—
[y]	yet	—
[w]	wet	—

Table: 5-3 English Phonetic Chart (After Falk, 1975)

2. A diphthong is a vowel sound in which the tongue starts in one position and then rapidly changes to another.

The chart below (table 5-4) illustrates how English consonants are distinguished by means of three phonetic features:

a) Manner of Articulation;
b) Place of Articulation;
c) Voicing (voiced [V] or unvoiced [U]).

Place / Manner	Labial		Labio-Dental		Dental		Alveolar		Palatal		Velar	
	V	U	V	U	V	U	V	U	V	U	V	U
Stops	*p*	*b*					*t*				*k*	*g*
Fricatives			*f*	*v*	*Θ*	*ð*	*s*	*z*	*š*	*ž*		
Affricatives									*č*	*ǰ*		
Nasals		*m*						*n*				*ŋ*

Table: 5-4 Phonetic features of Consonants

The stops, [b, p, t, d, k, g] are characterized by an abrupt release of air. The fricatives, [f, v, Θ, ð, s, z, š, ž] result in a friction-like noise. [m, n, ŋ] are nasal sounds because they are formed by movement of air through the nasal cavity. The affricatives [č, ǰ] and the fricatives [f, v, s, z, š, ž] are referred to as being strident because they are phonetically "harsher" than [Θ, ð].

Whorf's Formula

Numerous attempts have been made to formulate rules for the creation of an English word (Hawkins, 1984; Knopf, 1981; Malone, 1936; Newman, 1951; O'Connor & Trim, 1953). Benjamin Lee Whorf's formula is one of the oldest systems and describes the structure of a mono-syllable English word (Knopf, 1981). A syllable is defined as a sound combination containing a vowel (Falk, 1978). The word *reality* can be analyzed as:

[ri-æl-ə-ti].

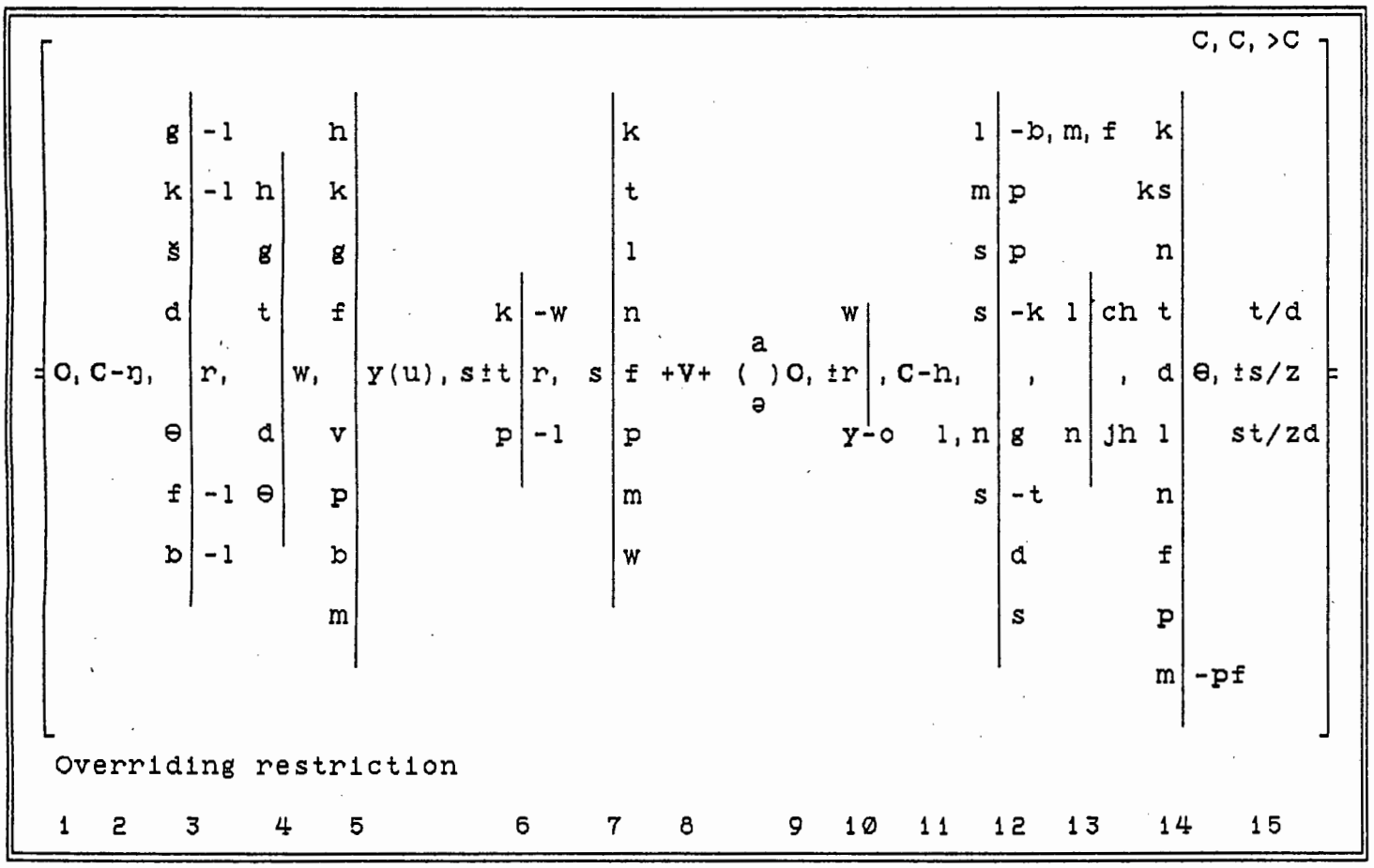


Table: 5-5 Whorf's Formula (after Knopf, 1981)

Figure 5-5 above illustrates Whorf's formula. The eighth term (+ V +) indicates that any English word must contain a vowel. The vowel can be preceded and followed by additional

phonemes. The Ø,C in the first term indicates that a word need not begin with a consonant (e.g. *off*). The C-ŋ in the second term excludes words beginning with /ŋ/, e.g. in *going*.

The third term consists of two columns of consonants, either side of a line, indicating that a word can start with /g, k, š, d, ϴ, f, b/ followed by /r/. (The greek 'theta', ϴ, denotes the *th* sound - as in *throw*). The consonants /g, k, f, b/ may also be followed by /l/. A word can start with the consonants /h, g, t, d, ϴ/, followed by /w/. Notice that the permissible consonant cluster /hw/ becomes *wh* in orthography. /ϴw/ is found in words such as [ϴwɔt] *thwart* and /gw/ is found in words such as [gwɛn] *Gwen*.

The fifth term allows for consonants, /h, k, g, f, v, p, b, m/, to be followed by /y/ only if followed by the vowel /u/. This combination is found in words such as [hyuw] *hue* and [fuyw] *few*. The sixth term allows a word to start with /k, t, p/ and may be preceded by /s/, but then if also followed by a consonant, this must be /r/ (e.g. *crew, screw; train, strain; pray and spray*). A word may be started with /k/ followed by /w/ (e.g. *squash*, where the "squ" is pronounced [skw]), or /p/ followed by /l/ (e.g. *split*). The seventh term allows words to start with /s/, followed by either /k, t, l, n, f, p, m, w/.

The first seven terms illustrate the permissible consonant clusters which may occur before a vowel (eighth term). The ninth term permits a word to end in a vowel as long as the vowel is *ah* (denoted by /a/) or the phonetic sound /ə/ which is found in the

word *the* when it is pronounced as *thuh*, rather than *thee*.

The tenth term shows that /w, r, y/ may follow a vowel, except when the use of one of these would result in joining /w/ and /y/ (indicated by the zero in the formula). The eleventh term allows a word to end in one consonant as long as it is not /h/ (in words such as *hallelujah*, the /h/ is not pronounced). Terms twelve, thirteen and fourteen are interpreted in the same way as are terms three, four and five. The fifteenth term can follow anything that occurs before it and describes the formation of plurals, such as *cats* ([ts]) or *dogs* ([gz]).

The overriding restriction, C,C,>C, eliminates words ending with the same consonant repeated. Although a word like *till* is spelled with two *l*'s, it is pronounced as only having one.

The validity of Whorf's formula can be expanded to include words of more than one syllable. For instance, the word [fətagrəfi], *photography* can be analyzed as follows:

Syllable	Analysis of Terms
/fə/	f - term 2; ə - term 8;
/tag/	t - term 2; a - term 8; g - term 11
/rə/	r - term 2; i - term 8
/fi/	f - term 2; i - term 8

Table: 5-6 Analysis of *photograph*

The Development of a Phonetic Rule Base

Sounds cannot randomly be combined to form words, but certain phonological conventions exist which dictate how words are formed (Gibson et al, 1962). In an attempt to augment the statistical anticipatory techniques in the previous chapter, these conventions will be used to formulate a series of phonetic rules to anticipate groups of sounds within a word.

A number of linguistic studies have investigated the phonological structure of words in terms of consonant clusters (O'Connor & Trim, 1953; Saporta, 1955; Carroll, 1938; Hawkins, 1985). The maximum number of consonants not separated by a vowel is three (Hawkins, 1984). Consonant clusters can occur initially, medially or finally.

In order to refer to the different consonant positions, the position of the individual consonants will be numbered from the vowel outwards, e.g.:

C C V C C	C C C V C C C
2 1 1 2	3 2 1 1 2 3
p l a n t	*s t r i n g s*

VOWELS:

Rule 1: Any vowel can begin a word except for [ʊ, ʊə] (O'Connor & Trim, 1953).

Rule 2: A vowel must follow the consonants [j, h, r, w].

Rule 3: If a diphthong is followed by a consonant cluster, V;C1;C2, the consonant in position C2 is will be alveolar.

Rule 4: Glides and nasals often follow a diphthong in position 1 of a consonant cluster.

WORD-INITIAL CONSONANTS:

Rule 5: A word may begin with any consonant, except for /ŋ/ and /ž/ (O'Connor & Trim, 1953).

Rule 6: Two-consonant clusters - C2;C1 word-initial position: (Hawkins, 1984).

6a: C1 will be an approximant /l, r, w, j/.

6b: C2 may be a stop /t, d, p, b, k, g/ or a fricative /f, θ/.

Rule 7: Two-consonant clusters with initial /s/ - /s/;C1 word-initial position (Hawkins, 1984): C1 will be a stop /t, d, p, b, k, g/, a voiceless non-strident fricative /š, ž/, a nasal /m, n/ or an approximant /l, w/. /r/ only occurs after /š/ (O'Connor & Trim, 1953).

Rule 8: Three-consonant clusters - C3; C2; C1 word-initial position (Hawkins, 1984) - table 5-7:

8a: C3 is /s/.

8b: C2 is voiceless stop /k, p, t/.

8c: C1 is an approximant /l, r, w/.

Position	Consonants
C3	s
C2	p t k
C1	l r w

Table: 5-7 Three consonant combinations

Rule 9: Three consonant clusters presuppose two consonant clusters (Hawkins, 1984), e.g. /stj-/ presupposes /st-/ and /tj-/; /spw-/ is not allowed as /pw/ does not occur).

WORD-FINAL CONSONANTS:

Rule 10: /e, æ, ʊ/ must be followed by a consonant (Hawkins, 1974).

Rule 11: Any single consonant may end a word except for /h, r, w, y/ [3].

3. Although these sound segments may occur at the end of words in the written language, they are not pronounced, e.g. *hoorah* = /hʊrə/; *may* = /me/.

Rule 12: For two-consonant clusters, word-final, the sounds may be analyzed as being in position C1 and C2 (arbitrary) or C2 and C3 (unambiguous in most cases) - table 5-8 below.

Segment position	V	C1	C2	C3	C4
Sound combinations		l m n ŋ s	any consonant except h r w y	θ t d s z	t (d) s

Table: 5-8 Three-consonant clusters word-final (After: O'Connor & Trim, 1953)

Rule 13: Three-consonant clusters word-final can be analyzed in positions: C1, C2 and C3; or C2, C3 and C4 (table 5-8, above). The choice is ambiguous, e.g. /l, m, n, ŋ + θs; l, n, s, k + ts; l, n, k + st/ as in *months; tents.*

Segment position	V	C1	C2	C3	C4
Sound combinations		k l m n	p t k s f	t θ s	t s

Table: 5-9 Four-consonant clusters word-final (After: O'Connor & Trim, 1953)

Rule 14: Only seven four-consonant clusters word-final (table 5.9 above) are found: /lkts/ as in *mulcts*, /mpst/ as in *glimpsed*, /ksts/ as in *texts*, /lfθs/ as in *twelfths*, /ksθs/ as in *sixths*, and /ntθs/ as in *thousandths*.

WORD-MEDIAL CONSONANTS:

Rule 15: Word-medial clusters are divided into word-initial and word-final syllables. E.g. the word /fiŋ.g/ is divided into a word-final /-ŋ/ (cf. ring) and a word-initial /g/ (cf. get).

COMPUTATIONAL IMPLICATIONS OF THE PHONETIC RULE BASE

The identification of four groups of letters (fig. 5-10) allows for a degree of anticipation using a static display. The first group consists of the consonants, /b, c, d, f, g, k, l, p, s, t/; the second group includes the nasals, /m, n/, the glides, /h, r/ and the semivowels /v, y/; the third group contains the vowels, /a, e, i, o, u/; and the fourth group contains /w, x, y, z, q/. /s/ occurs in each class as it can occur in all consonant positions. The groups were chosen to approximate phonetic classes and combination structures.

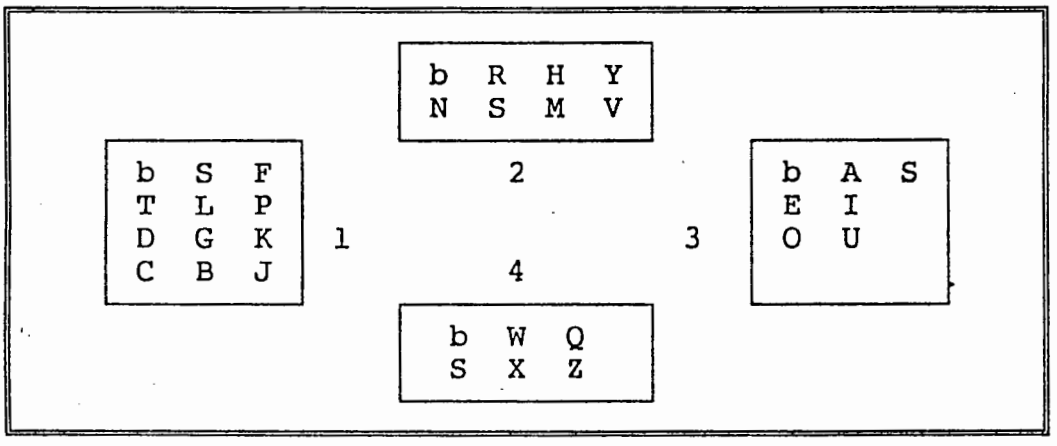


Fig: 5-10 The layout of the linguistic groupings (where "b" denotes the space).

Instead of scanning one matrix, the system scans the four groups according to linguistic letter order. Once a group has been selected, the individual matrix is scanned. In this way, cognizance is taken of position within the word. However, the user is always aware of the location of the target symbol and no additional cognitive overhead is required to ascertain location before scanning can begin. The letters within the groups are arranged according to global letter frequencies.

In order to use these groups effectively, the rules identified in the preceding discussion were formalized and a rule-base constructed (figure 5-11).

Linguistic Rule	Derived from:
1. Begin in word-initial, scan group 1. Reason: Beginning of word.	Rule 5
2. If <space> go to step 1. Reason: End of word.	
3. If /q/ chosen, add /u/. Reason: Syntax.	
4. If /s/ chosen, scan group again. Reason: Consonant clusters.	Rules 5, 7, 8
5. If /t, p, k/ chosen or no match, scan group 2. Reason: Consonant clusters.	Rule 6
6. If word-medial, go to step 1. Reason: Word-medial -> group 2 -> 1.	Rule 15
7. Scan group 3. Reason: Vowel segment.	Rule 5
8. If /a, e, i, o, u/ chosen, scan again. Reason: Diphthongs.	
9. If /w, q/ chosen, scan group 3. Reason: Vowel required.	Rule 3
10. If /s, r, m, n/ chosen and word-medial, scan group 2 again. Reason: Double consonants in orthography.	
11. Scan group 2, Go to step 6. Reason: Word-medial or word-final. (Word-final handled as word-initial).	Rule 15
12. If no match, scan hierarchically: 1-2-3-4. Reason: Exceptions to rules and group 4.	

Fig: 5-11 Rule Base for linguistic anticipatory system

The cost matrix of each group is shown in figure 5-12:

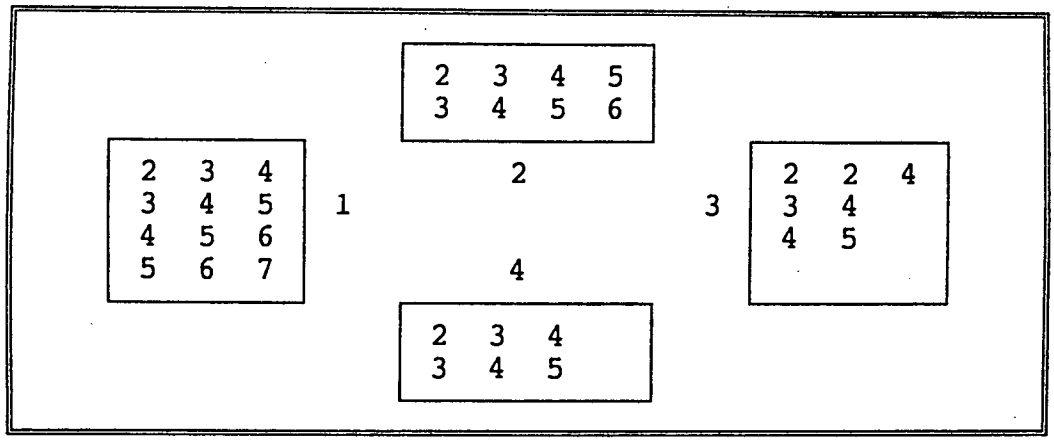


Fig: 5.12 The cost matrix for the linguistic groupings.

A drawback of the system is that three switch activations are needed to select a specific target letter, in contrast with the two needed for the row-column scanner. Therefore, the cost (wi) of selecting an item has an additional cost as the group has to be selected first. This means that the actual cost is increased as follows:

```
w(i) = c(i) + 1     if the first group is chosen;
     = c(i) + 2     if the second group is chosen;
     = c(i) + 3     if the third group is chosen;
     = c(i) + 4     if the fourth group is chosen.
```

The best case is thus: $w(i) = c(i) + 1$;

and the worst case: $w(i) = c(i) + 4$.

The communication efficiency, J, is defined as:

$$J = \sum_{i=1}^{Nj} P(i)\, w(i)$$

where Nj = the number of letters in the jth group;
P(i) = the probability of the ith letter;
w(i) = the cost of the ith letter.

The probabilities for individual letters and the calculation of system efficiency can be seen in table 5-13.

Letter	P(i) x 10^3	c(i)	P(i) [c(i) + k] k=1	k=2	k=3	k=4
space	174	2	522	696	870	1044
E	108	3	432	540	648	756
T	87	3	348	435	522	609
A	67	3	268	335	402	469
O	66	4	330	396	462	528
N	59	3	236	295	354	413
R	56	3	224	280	336	392
I	52	4	260	312	364	416
S	50	3	200	250	300	350
H	43	4	215	258	301	344
D	31	4	155	186	217	248
L	28	4	140	168	196	224
F	24	4	120	144	168	192
C	23	5	138	161	184	207
M	21	5	126	147	168	189
U	20	5	120	140	160	180
G	16	5	96	112	128	144
Y	16	5	96	112	128	144
P	16	5	96	112	128	144
W	13	3	52	65	78	91
B	12	6	84	96	108	120
V	8	6	56	64	72	80
K	2	6	14	16	18	20
X	1	4	5	6	7	8
J	1	7	8	9	10	11
Q	1	4	5	6	7	8
Z	1	5	6	7	8	9
$\sum_{j=1}^{4} \sum_{i=1}^{Nj} P(i)\, w(i) =$			4352	5348	6344	7340

Table: 5-13 Letter probabilities and the number of cursor-steps required to select a character in the linguistic display.

For the best case in a four-group system, the efficiency is thus:

$$J = \sum_{j=1}^{4} \sum_{i=1}^{Nj} P(i)\, [c(i) + 1]$$

$$= 4.352$$

and in the worst case, the efficiency is thus:

$$J = \sum_{j=1}^{4} \sum_{i=1}^{Nj} P(i)\,[c(i) + 4]$$

$$= 7.340$$

If these letter groups were to be scanned in a random manner, the efficiency rate would not be very high and no advantage would be gained from the investigation into word structure.

It is obvious that the more groups scanned, the worse the efficiency would be. The aim should therefore be to anticipate the correct group, thus limiting the actual cost to:

$$w(i) = c(i) + 1$$

The finite state diagram (figure 5-14) summarizes the rule-base:

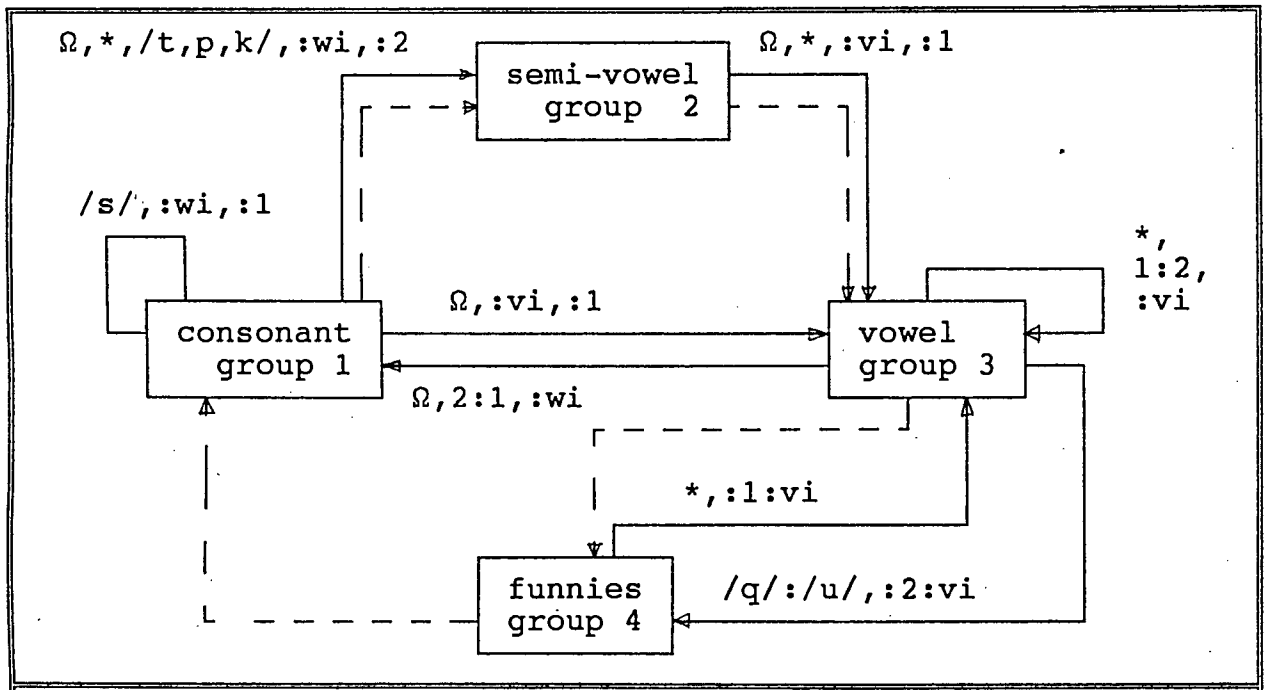


Fig: 5-14 Finite state diagram of rule base

CHAPTER SIX

SOFTWARE DEVELOPMENT

INTRODUCTION

Much discussion has been directed to different methods of improving the wordprocessing speed when a scanning technique is implemented. These methods range from static displays in which a letter matrix is scanned to a dynamic word prediction system. A new method which utilises both linguistic and statistical knowledge was introduced in the previous chapter.

All of these theories attempt to allow a severely disabled user to produce written language in the most efficient way possible. The characteristics of an efficient system were discussed in chapter 4. Because of the number of different theories, it is necessary to have a benchmark by which different scanning systems and philosophies can be analyzed and compared.

In order to achieve a comparative analysis, the different theories would have to be to computerized. However, each theory requires an individual implementation and to design separate programs would be too time consuming and inflexible.

A software system which could accept different systems in a straight forward format, allow for interactive manipulation by the user and result in a uniform quantitative analysis was thus devised.

CHOICE OF PROGRAMMING LANGUAGE

Anticipatory wordprocessors for the disabled have been implemented on microcomputers using traditional programming languages, including BASIC, Fortran and Pascal (Poon, 1983; Swiffen, et al, 1985). These languages are appropriate when using the statistical and/or global approaches which underly the anticipatory nature of these programs. Because of the linguistic nature of ideas introduced in the previous chapter, it was decided that a programming language, more suited to natural language manipulation should be considered.

The LISP programming language was chosen for implementation for the following reasons:

- LISP, compared with other symbol manipulation programming languages, is widely used for natural language processing (Winston and Horn, 1984), whereas PROLOG, another possibility, is used mainly in the implementation of expert systems.
- In addition, the interactive and modular attributes of LISP proved essential in the development of the new system. When programming in languages such as Pascal, the programmer is restricted by the necessity for re-compilation each time the program is modified. LISP, however, allows for re-compilation of individual functions within the system. The programmer thus has far more freedom when interactive experimentation during program execution is required.

HARDWARE AND SOFTWARE REQUIREMENTS

Two LISP development systems were considered. Golden Common LISP (Gold Hill Computers, 1985) was chosen above IQLISP (Integral Quality, 1984) because of its universal acceptance. Common Lisp is widely used and much of the literature on LISP uses Common LISP in examples, which made it easier to grasp the basic constructs of the language.

Golden Common Lisp was installed on an IBM-compatible personal computer with 640K of memory and a 20 megabyte fixed disk. The GCLISP development system comprises code residing on five 5, inch floppy diskettes and requiring at least 512K of memory. Because of the demands on the system, it was felt that a fixed disk would be needed for efficient software development.

Because of its operating overhead, LISP is not suitable for implementing a wordprocessing package. However, LISP provides an ideal research environment in which theories can be implemented, verified, analyzed and refined. The reason for developing the software was to provide tools to evaluate the performance of writing devices.

The software which has been developed is therefore not meant to be a commercially available wordprocessor, but is rather a research tool to analyze different scanning philosophies.

SYSTEM OVERVIEW

In order to construct, experiment with and compare several scanning procedures, a general purpose program was developed which accepts any scanning system and runs it in real time. The layout of linguistic item groups and rule-base is specified in a predefined syntax acceptable to the software.

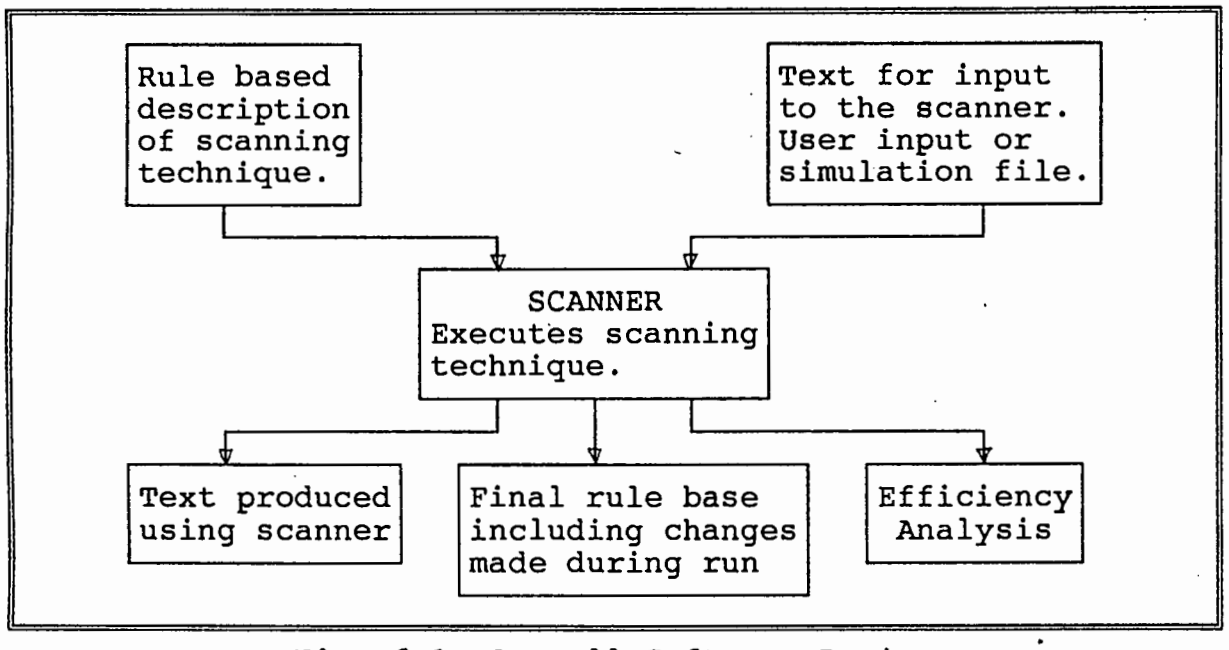


Fig: 6-1 Overall Software Design

There is no limitation on what the linguistic items should be. It is thus possible to use any structural element, be it a graphic symbol, letter, a morpheme, a word, a phrase or a sentence.

Once this description has been read, the system can be used either interactively by the user, or in simulation mode during which the system accepts a benchmark passage, which is reproduced using the scanning system.

At the end of each session, a statistical report documenting characteristics and performance of the scanning procedure is stored for future analysis. The "automatic" simulator makes no allowance for typical user-errors such as typographical errors, and inaccurate or slow switch activation. The statistical cost-analysis is thus a best-case scenario, which in practice, would be degraded by cognitive and physical factors.

PROGRAM SPECIFICATIONS

The actual program specifications are as follows:

Input Specifications

1. A description file containing specifications for individual scanning procedure;
2. A one-switch input to be used to select options on the screen;
3. A text file which identifies the output required when a program simulation is specified;
4. Interactive modification of the rule base is to be supported via the keyboard.

Processing Specifications

1. The description file containing scanning procedure specifications must result in the following actions:
   * identification of scanning procedure;
   * identification of base delay factor for scanning;
   * creation of linguistic item groupings;
   * creation of the corresponding rule base.

2. The software must proceed through the rule base until a rule antecedent is found which matches the current state of the environment. The corresponding consequent is then processed and the system must await a system interrupt which is characterized in three ways:
   * in the event of interactive usage, the interrupt will be the activation of the external switch;
   * in the event of a simulation, the software will note the subsequent item in the text file and initiate an interrupt as soon as the required item is scanned on the monitor;
   * in order to modify the rule base, the user must be able to interrupt the scanning process interactively while the program is running.
3. The software will prompt the user for:
   * a name which will identify the run;
   * a text parameter file containing a description of the scanning system to be implemented;
   * a text file containing a passage of text to be used for simulation purposes.

Output Requirements

1. Interactive feedback of text production.
2. Interactive interrogation of the rule base.
3. A statistical report to be compiled must include:
   * every item chosen by the user;
   * the actual text produced by the scanning system;
   * an analysis of the scanning rate;
   * the modified rule base.

SOFTWARE DESIGN

The transcription of a scanning system into a set of rules which can be interpreted by the evaluator, requires a knowledge of LISP. This knowledge also enables the user to manipulate the scanning environment during program execution. Procedures can be created or modified easily, while retaining the core program which acts as an interpreter for the scanning mechanism. The interactive nature of the software is illustrated by the flowchart in figure 6-2.

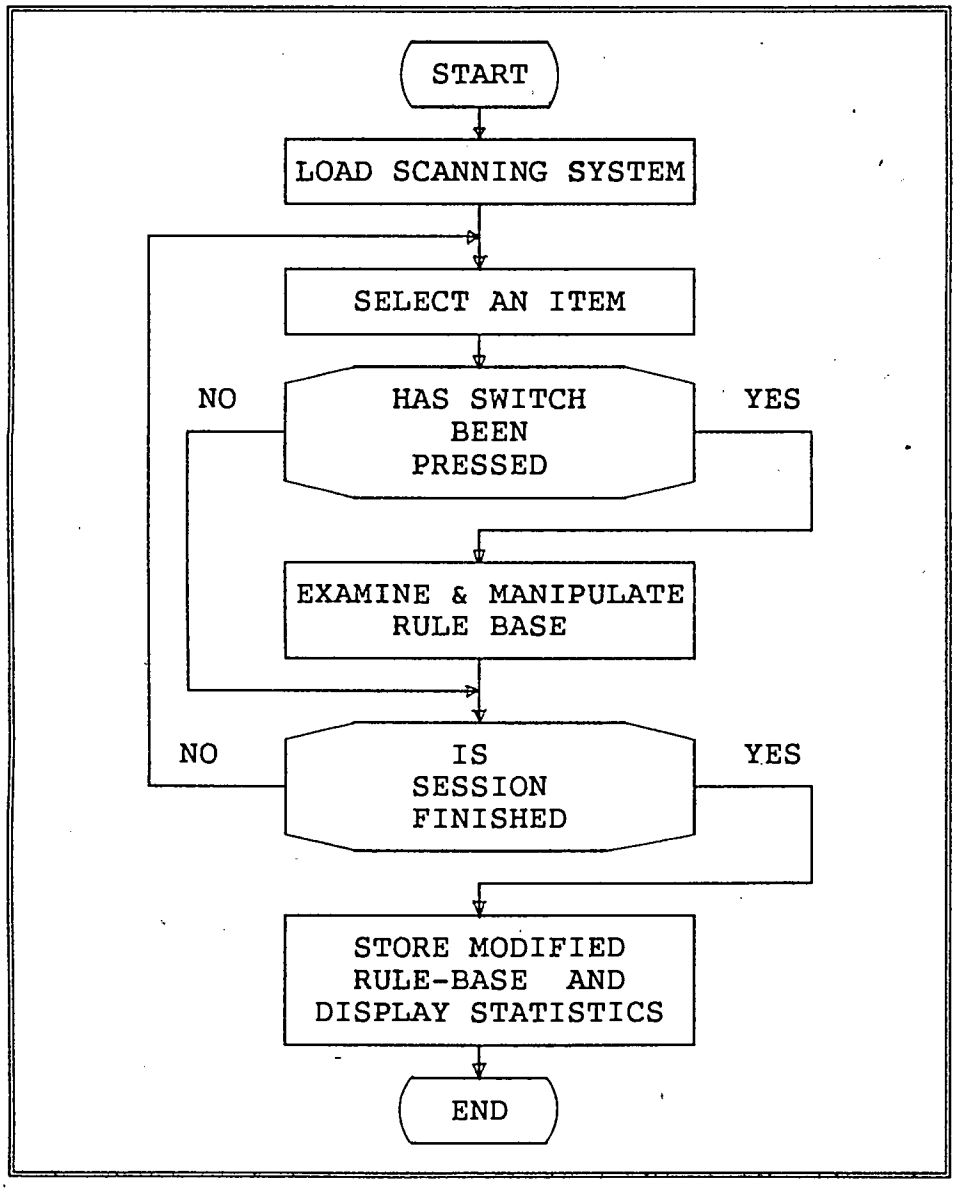


Figure 6-2: Flowchart of system mechanism.

A detailed description of the design of the program can be found in appendix 2.

DESIGN OF THE RULE-BASE

A LISP command file is needed to specify, initiate and control a scanning procedure. The command file must contain the following sections:

* an identification section which specifies the name of the scanning system and a delay factor;
* the structure of the linguistic item groups;
* definitions of any functions needed to execute the scanning system;
* a rule-base which will control the scanning procedure;
* the initialization of the system.

Documented command files for the scanning systems used in chapter 7 are listed in appendix 3.

PROGRAM EXECUTION

The program is executed from within the LISP environment, thus giving the user full use of the LISP's interactive facilities. The program begins by asking for a run name and a scanning command file (Default: Static Statistical System).

Figure 6-3 lists the execution modes and their functions:

On (Yes)	Function	Off (No)
Interactive Interactive	Rule-Base Interrogation Scanning Control	Disabled Simulation

Fig: 6-3 Execution Modes

The rule-base interrogation mode allows the user to view, modify, add rules to, or delete rules from the rule-base. The user is given the opportunity to interrogate the system whenever a rule is evaluated.

If the program is in the simulation mode, a text file is used as input. The software simulates the switch activations needed to reproduce the text. The user can suspend the execution for rule-base modification if interrogation mode is on. Execution is halted when all text from the simulation text file has been reproduced.

In interactive scanning mode, the user can press any keyboard key to indicate a switch activation and is able to halt the execution by selecting the "quit" symbol, Ω.

An efficiency report is displayed when the scanning procedure is finished and may be printed. This report and all other output is stored in files which are identified by the name of the run. An example of such a report is shown in figure 6.4 on the following page.

The user has two options at the end of execution: a) exiting from the program or b) restarting the system.

```
            COMMUNICATION RATE AND EFFICIENCY REPORT
            ========================================

Anticipatory Algorithm:  "Static Statistical"

Simulation File:         PARA3.TST

Date: 19 AUGUST 1988     Duration:     0 hours 18 minutes

                         Delay factor: 0

Processed Text to be found in file called:     STAT3.NEW

Mirrored Text to be found in file called:      STAT3.MIR

Modified rule base to be found in file called: STAT3.RUL

Number of elements chosen: 778

Number of deletions:          0 Deletions per element:  0
                         ------

    Total chosen:           778


Number of selections:      1556 Selections per element: 2

Scans not selected:        2014 No    "    per element: 2.59
                         ------

    Total scan:            3569 No Scan / scan        : 0.78

Observed Efficiency (O)    4.59 scans per element

Expected Efficiency (E)    4.47

% Difference (O - E)/E     0.03
```

Fig: 6-4 Example of statistical report for analysis.

CHAPTER SEVEN

COMPARISON OF SYSTEMS

INTRODUCTION

In order to test and quantify the efficiency of various linguistic group-based systems, a standard series of passages were presented as input to these different systems. The results obtained from these experiments were documented and analyzed.

SCANNING SYSTEMS

Six different scanning systems, three of which are currently in use were selected (table 7-1). Each system was transcribed into a series of rules compatible with the evaluation system. The different rule bases are listed in appendix 3.

Name	Type of Scanning System	Origin
StatA	Static statistical	Poon, 1983
StatB	Static statistical	Clarke, 1987
Digram	Dynamic statistical	Gibler, 1981
LingA	Static linguistic	See table 7-2
LingB	Static linguistic	See table 7-2
LingC	Static linguistic	See table 7-2

Table: 7-1 List of Scanning Systems and their origins

The first three systems are currently being used in practice and have been referred to in previous chapters. The static

linguistic systems are novel implementations of the linguistic theory discussed in chapter 5. The three variations of the statistic linguistic systems have slightly different rule bases to illustrate the savings which occur when additional rules are incorporated (table 7-2).

Name	Description of Additional Rules
LingA	Linguistic groups without rules, i.e. hierarchical scanning
LingB	Linguistic groups with a rule base
LingC	LingB + diphthong scan

Table: 7-2 Variations of the linguistic system

All systems have been implemented solely as upper-case systems without punctuation, as the efficiency of word construction, and not sentence construction, was under investigation.

THE EXPERIMENTS

Five standard passages were used as input to the various scanning procedures. These passages are identical to those used by Gibler (1981) in order to provide a benchmark between the systems. The sources for the passages include a composition by a writer who is disabled, books, magazines and newspapers and were chosen as a "typical" representation of English text. The source, text, and number of letters, of each passage follow.

Experiment 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 QUANINE 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

The analysis of the passage is as follows:

Source of Passage: MARSHALL R. (ed.) 1977. Reader's Digest Great Events of the 20th Century. The Reader's Digest Association, Inc., 454-461.
Passage Name: PARA1.TST
Number of letters: 767

Experiment 2:

AT THE HEARING GERALD HIS MOTHER OLDER BROTHER AND TWO PROBATION OFFICERS APPEARED BEFORE THE JUVENILE COURT JUDGE GERALD S FATHER WAS OUT OF TOWN ON BUSINESS THE COMPLAINING NEIGHBOR 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 DIALED MRS COOK S 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 RELEASE 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

The analysis of the passage is as follows:

Source of Passage: WEINGARTEN P. 1981. "Children's Rights: How Old Is Old Enough?", Chicago Tribune, Sunday, 15 February.
Passage Name: PARA2.TST
Number of letters: 796

Experiment 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 HOBBIES 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

The analysis of the passage is as follows:

Source of Passage: KENNAN G.G. 1981. "A Diplomat Warns That The 11th Hour Is Upon Us", Chicago Tribune, Sunday, 15 February.
Passage Name: PARA3.TST
Number of letters: 778

Experiment 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 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

The analysis of the passage is as follows:

Source of Passage: PFROMMER M. Composition by a disabled writer.
Passage Name: PARA4.TST
Number of letters: 883

Experiment 5:

WITH THE ADVENT OF THE INDUSTRIAL REVOLUTION HOWEVER WORK MOVED OUTSIDE THE HOME BECAME SEGMENTED AND LOST MUCH OF ITS INTRINSIC VALUE IN THE POST WORLD WAR II ERA WORK TOOK ON EVEN MORE OF AN INSTRUMENTAL ORIENTATION TODAY THE QUESTION IS EVEN RAISED IF WORK IS A CENTRAL LIFE INTEREST FOR A MAJORITY OF THE WORK FORCE REGARDLESS OF THE ROLE WORK PLAYS IN LIFE TO THE PHILOSOPHER OR TO THE THEOLOGIAN THE DIGNITY OF MAN MUST BE OF PRIMARY CONCERN IN ITS DESIGN HUMAN BEINGS ARE DESERVING OF DIGNITY IN AND OF THEMSELVES TO JOHN JULIAN RYAN THE AUTHOR OF THE FIRST ARTICLE IN THIS SECTION A TRULY HUMANE SOCIETY IS ONE IN WHICH THE PRIMAL NEEDS AND RIGHTS OF EVER MAN AS A FULL HUMAN BEING ARE RESPECTED AS SACRED NOT THE LEAST OF THESE BEING THAT OF LEADING A MEANINGFUL CONSCIOUSLY CREATIVE LIFE AS A WORKER SERVING OTHERS SKILLFULLY PERSONALLY AND HONORABLY YET RYAN CAUTIONS WHILE WORK CANNOT BE LOOKED UPON MERELY AS A OPPORTUNITY TO MAKE MONEY NEITHER CAN IT BE REGARDED PRIMARILY AS A MEANS OF SELF

The analysis of the passage is as follows:

Source of Passage: HEISLER J.W., HOUCK J.W. 1977. (Eds.) A Matter of Dignity, Inquiries into the Humanization of Work, University of Notre Dame Press, 7-8.
Passage Name: PARA5.TST
Number of letters: 1005

DISCUSSION OF RESULTS

Switch Activations

Referring to table 7.3 on the following page, systems which have a static display need a constant number of activations to select a letter, i.e. 2 selections for a row-column system and 3 for a group-row-column system. However, this number varies in the dynamic digram system - if the desired letter is offered in the first display, only 1 selection is needed; if not, further selections are needed to display subsequent options. The dynamic system requires the least number of selections (1.35 on average).

Parameter	Exp	StatA	StatB	Digram	LingA	LingB	LingC
Number of Switch Activations	1	1532	1532	1078	2332	2326	2326
	2	1592	1592	1076	2421	2418	2799
	3	1556	1556	988	2365	2362	2362
	4	1764	1764	1218	2680	2674	2680
	5	2010	2010	1315	3050	3047	3047
Average		1691	1691	1135	2570	2565	2643
Number of Selections per Letter	1	2	2	1.41	3	3	3
	2	2	2	1.36	3	3	3
	3	2	2	1.27	3	3	3
	4	2	2	1.38	3	3	3
	5	2	2	1.31	3	3	3
Average		2	2	1.35	3	3	3

Table: 7.3 Number of Switch Activations in Experiments 1 to 5

Redundancy

The number of irrelevant scans [1], are lowest in the linguistic systems (table 7.4 on the following page). This is because the linguistic systems first consider "legitimate" occurrences, whereas the digram system does not. Comparison between the three linguistic systems reveals that irrelevant scans are decreased as appropriate linguistic rules are added. "LingA" has no knowledge base and does not attempt to anticipate the next group. "LingA" therefore shows the most irrelevant scans. "LingB" has a better performance in terms of the number of irrelevant scans. "LingC", has one additional rule which allows for a two-vowel (dipthong) occurrence in the middle of a word. However, instead of improving the system, the inclusion of this rule places too many restrictions on the system and actually degrades its performance.

1. An irrelevant scan is a scan which is not selected because the target letter does not appear in set of letters currently under consideration.

The "StatA" system has less irrelevant scans than the digram system, whereas the "StatB" system ignores the most scans. The digram system demands far more irrelevant scans if the required letter happens not to occur in the first display. The advantage of positioning of letters within the "StatA" system according to statistical occurrence can be seen when compared to the "StatB" system (a less structured system) which exhibits more irrelevant scans than the "StatA" system.

Parameter	Exp	StatA	StatB	Digram	LingA	LingB	LingC
Number of Ignored Scans	1	1944	2430	2619	1877	1651	1749
	2	2019	2565	2353	1918	1681	2050
	3	2014	2456	2000	1979	1756	1865
	4	2308	2875	2825	2155	1916	2014
	5	2569	3176	2804	2471	2149	2281
Average		2171	2700	2520	2080	1831	1992
Total Number of Scans	1	3475	3962	3696	4208	3976	4074
	2	3610	4159	3428	4338	4098	4848
	3	3569	4012	2987	4343	4117	4226
	4	4071	4639	4042	4834	4589	4693
	5	4578	5186	4118	5521	5195	5327
Average		3861	4392	3654	4649	4395	4634
Irrelevant Scans / Total Number of Scans	1	0.56	0.61	0.71	0.45	0.42	0.43
	2	0.60	0.62	0.69	0.44	0.41	0.42
	3	0.56	0.61	0.67	0.46	0.43	0.44
	4	0.57	0.62	0.70	0.45	0.42	0.43
	5	0.56	0.61	0.68	0.45	0.41	0.43
Average		0.57	0.61	0.69	0.45	0.41	0.43

Table: 7.4 Irrelevant Scans in Experiments 1 to 5

It is interesting to note the ratio of irrelevant scans to total scans (table 7.4) as a **measure of redundancy.** The less time spent on irrelevant scans, the smaller the redundancy and the closer this ratio approaches zero. The aim of the digram system is to offer the desired next element in the first display.

However, in practice, the digram system produces a redundancy ratio of 0.69, the worst of all five systems. This is probably due to the fact that to choose the next list of elements, five letters have to be scanned before choosing the asterisk, thereby requesting the next five letter display. Linguistic systems, by contrast, tend to offer less "impossible" options and ratios are thus closer to zero.

Efficiency

A superficial glance at the efficiency, i.e. the number of scans per element, suggests that the digram system is the most efficient (table 7-5). This observation corroborates the findings of Gibler (1981).

Parameter	Exp	StatA	StatB	Digram	LingA	LingB	LingC
Observed Efficiency	1	4.54	5.17	4.83	5.49	5.19	5.32
	2	4.54	5.21	4.31	5.46	5.15	5.27
	3	4.59	5.16	3.85	5.59	5.30	5.44
	4	4.62	5.25	4.58	5.48	5.20	5.31
	5	4.56	5.16	4.10	5.49	5.18	5.31
Average		4.57	5.19	4.33	5.50	5.20	5.33
Expected Efficiency		4.47	5.08	3.60	7.34	4.40	4.40
Percentage Difference		-2%	-2%	-20%	25%	-18%	-21%

Table: 7.5 Efficiency Results in Experiments 1 to 5

The relative differences between observed and expected efficiency are very slight in the static statistical systems ("StatA" and "StatB"). However, these differences are far more obvious in anticipatory systems due to the character of natural

language. The observed efficiency of the digram system is 20% worse than the predicted value, whereas the "LingA" system exceeded its predicted value by 25%. This improvement is because the expected value is always calculated by assuming the worst case. Any serendipitous anticipation was therefore not "expected" and reduced the number of scans needed for that letter selection accordingly.

A necessary characteristic of an effective writing system is efficiency (as defined on page 61). If this were the only criterion, the digram system would be by far the most effective. However, the predictions offered by such a system are not always correct and up to twice as many irrelevant scans as selected scans occur (table 7-4).

The three linguistically based systems, "LingA", "LingB" and "LingC", exhibit the more favourable redundancy ratios: 0.45; 0.41; and 0.45 respectively, compared to those calculated for the "StatA" (0.57), "StatB" (0.61) and digram (0.69) systems.

In contrast, the observed efficiencies for the linguistic systems vary between 5.2 to 5.5 scans per letter, compared with the more efficient 4.3 observed in the digram system. Although the additional cost is at most one scan per letter (the difference between 5.2 and 4.3), the reduction in the number of irrelevant scans using the linguistic approach more than offsets this cost.

IMPLICATIONS FOR ANTICIPATORY SYSTEMS

It has been shown that although statistical information regarding letter usage is important in the implementation of an efficient writing system, additional linguistic information can significantly reduce the amount of irrelevant scanning. This additional information does not require large memory storage as do statistical tables, but relies rather upon programmed rule-based manipulations. The implementation of some linguistic knowledge in a statistical system will therefore not impose the same large additional memory requirements needed to store additional statistical data.

The adoption of a static form of letter anticipation matrix is of interest when trying to minimize the search time and corresponding cognitive load experienced by the system user. However, the increased number of switch activations (1.35 for row digrams 2 for row/columns, and 3 for groups) may in practice outweigh reductions in irrelevant scans. Detailed movement time studies should be undertaken to measure and compare the overall performance of different systems in practice. By including linguistic attributes, digram systems might show in still further improvement.

The preceding investigation concentrated specifically on letter anticipation. It has been suggested by Gibler (1981) that the implementation of word vocabulary lists further increases the efficiency of a writing system. Research into sentence parsing is advanced and the results obtained when defining rules for the letter structure of words indicate that an investigation into the syntactic and semantic relationships of words may similarly increase efficiency by vocabulary list ordering.

CHAPTER EIGHT

CONCLUSION

The development of a universal evaluation system has provided an objective method by which different communication strategies can be compared. In this present work, several systems were transcribed into rule-bases and were then evaluated and analyzed.

One of the systems which was evaluated was based on phonological rules which govern the structure of English words. Identification of relevant phonological rules resulted in the design of an anticipatory system using a static display of five groups of letters. The selection of these groups was successfully anticipated when the preceding letters and their relative position within a word were analyzed using linguistically based rules.

Although the traditional measure of communication efficiency did not improve on the digram system, a new measure of redundancy showed that less irrelevant scans were offered when the linguistically based systems were used.

It also appears that a larger cognitive load is imposed by dynamic matrix layouts due to the increase in visual search time. The design of the linguistically based system eliminates dynamic matrices, but succeeds in retaining a degree of anticipation. Although the initially observed efficiency rate was lower that that of the tested dynamic systems, it increased when new rules were added to the rule base. The overall reduction of cognitive

load should however improve this rate. This hypothesis needs to be investigated further.

IMPLICATIONS FOR FUTURE RESEARCH

The research discussed in this dissertation has concentrated on letters as linguistic elements. It has been shown by others, however, that the implementation of word-lists, in the form of dictionaries, can improve the communication rate ten-fold (Gibler, 1981).

A similar linguistic approach applied to the anticipation of words, as opposed to letters, promises to improve on simple alphabetic listings of word menus. For example, examine the following partial sentence: "The man sat ...". It is intuitive to predict that the next word might be a preposition (e.g. "The man sat on ..."); or an adverb (e.g. "The man sat slowly ..."); or an object (e.g. "The man sat the child ...). By noting the position within sentences (i.e. parsing the sentence), the list of words offered to a disabled typist can be reduced even further by selecting not only statistical probabilities, but also by applying linguistic syntactic and semantic rules in the anticipatory procedure.

REFERENCES

All references cited in the dissertation are listed alphabetically. For additional background information consult the reading list in appendix 5.

ARDITI A., GILLMAN A.E.
1986
Computing for the blind user. Byte, March, 199-208.

ARNOTT J.L.
1987
A comparison of palantype and stenograph keyboards in high-speech output systems. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 106-108.

BAKER B.
1982
'Minspeak'. Byte, 7:2,, 186-202.

BAKER B.
1984
Chopsticks and Beethoven. Communication Outlook, 6:1, 8-10.

BAKER B.
1985
Communication mapping for semantic compaction systems. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, 123-125.

BAKER B.
1987
Semantic compaction for sub-sentence vocabulary units compared to other encoding and prediction systems. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 118-120.

BARKER M.R.
1987
A strategy and tools to train augmentative communication skills with single switch users. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 160-161.

BATES P.
1985
New developments in handicapped access. Creative Computing, 11:3, 124-127.

BENTRUP J.A.
1987
Exploiting word frequencies and their sequential dependencies. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 121-123.

BELETSA G.S.
1977
Anticipatory communication. A thesis. Tufts University, 2-227.

BOONZAIER D.A., KLEVIANSKY M.
1987
A 'smart' microprocessor-based infra-red data link to enable wireless wheelchair-to-'landbase' communication with personal computer and environmental control devices. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 691-694.

BOURNE C., FORD D.
1961
A study of the statistics of letters in English words. Information and Control, 4, 48-67.

CHILDRESS D.S., HECKATHORNE C.W., GIBLER C.D.
Human performance models and other issues in the design of anticipatory scanning communication aids. In Northwestern University Annual Report, 88-89.

CLARKE G.
1987
A user's perspective. Interface Newsletter 1:3, 4.

CREECH R.
1984
The key that releases the soul of man. In Conversations with Non-Speaking People. Toronto: Canadian Rehabilitation Council for the Disabled, 51-56.

DAMPER R.I.
1984
Text composition by the physically disabled : A rate prediction model for scanning input. Applied Ergonomics, 15:4, 289-296.

DAMPER R.I.
1986
Rapid message composition for large vocabulary speech output aids: A review of the possibilities. Augmentative and Alternate Communication, 2:4, 152-159.

DAMPER R.I., SMITH J.W., DABBAGH H.H.
1986
Shortforms: abbreviated natural language for computer text entry. Manuscript to be submitted to IEEE Transactions on Systems, Man and Cybernetics.

DAMPER R.I., BAKER R.G., LAMBOURNE A.D., DOWNTON A.C., KING R.W., NEWELL A.F.
1979
Educational subtitling for deaf children. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 304-305.

DEMASCO P., HORSTMANN H.
1987
A line of gaze communication system. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 112-114.

DENES P.B.
1963
On the statistics of spoken English. Journal of the Acoustical Society of America, 35, 892-904.

DEROOST G.
1986
Een Nederlands communcatiesysteem voor nietverbale personen gebaseerd op frekwente grafeemsequenties. Katholieke Universiteit Leuven, Factulteit Geneeskunde, Afdeling Logopedie, 1-77.

DOLBY J., RESNIKOFF H.
1964
On the structure of written English words. Language, 40, 167-196.

FALK J.S.
1978
Linguistics and Language. John Wiley & Sons, Inc.

FANG I.E.
1966
It isn't ETAOIN SHRDLU; It's ETAONI RSHDLC. Journalism Quarterly, 43, 761-762.

FANT A.
1982
Braille writing in Pascal. Byte, 7:9.

FOULDS R., SOEDE M., VAN BALKOM H., BOVES L.
1987
Lexical prediction techniques applied to reduce motor requirements for augmentative communication. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 115-117.

FOWLER M.
1957
Herdan's statistical parameter and the frequency of English phonemes. In E. Pulgram (ed.), Studies Presented to Joshua Whatmough, Gravenhage: Mouton & Co., 47-52.

FRAPRIE F.
1950
Frequency of letters in English. In H. Hansen (ed.), World Almanac and Book of Facts for 1950, N.Y.: New York World-Telegram, p. 543.

FRENCH J.J., KIRBY N.A., SIEBENS A.A.
1987
A rugged vane-type breath actuated call switch. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 145-147.

GERBER S., VERTIN S.
1969
Comparative frequency counts of English phonemes. Phonetica, 19, 133-141.

GIBLER C.D.
1981
Linguistic and Human Performance Considerations in the Design of an Anticipatory Communication Aid. Ph.D. dissertation, Northwestern University.

GIBLER C.D., CHILDRESS D.S.
1982
Language anticipation with a computer based scanning communication aid. Proceedings of the IEEE Computer Society Workshop on "Computing to Aid the Handicapped", Charlottesville, Vir., 11-16

GIBSON E.J., PICK A.D., OSSER H., HAMMOND M.
1962
The role of grapheme - phoneme correspondence in the perception of words. American Journal of Psychology, 75, 554-570.

GOLD HILL COMPUTERS
1985
Golden Common Lisp - Reference Manual. 163 Harvard Street, Cambridge, MA2139.

GROSSNER C.P., RADHAKRISHNAN T., POSPIECH A.
1983
An integrated workstation for the visually handicapped. IEEE Micro, June, 8.

HAWKINS P.
1984
Phonemes in sequence. In: Introducing Phonology. Hutchinson, 50-72.

HECKATHORNE C.W.
1985
Dynamic displays and predictive communication aids. 38th ACEMB, Chicago, Ill., 308-309.

HECKATHORNE C.W.
1986
Applying switches for control of augmentative devices. RESNA '86 Instructional Course, Minnesota, Mn.

HECKATHORNE C.W., CHILDRESS D.S.
1983
Applying anticipatory text selection in a writing aid for people with severe motor impairment. IEEE Micro, June, 17-23.

HILL L., BENNETT R., PISTELL D.
1987
Communicating with speaking-disabled students in mainstream educational settings. Communication Outlook, 8:1, 23-35.

HOYT R.K.
1986
Hearing impairment and micro-computers. Communication Outlook, 7:2, 12-13.

INTEGRAL QUALITY
1984
IQLISP - Reference Manual.

JAROS L.A., LEVINE S.P., PANCIOLI A.M.
1987
Altkey: a special inputs program for the IBM-PC. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 714-716.

JOHNSON R.
1985
PCS. Communicating Together, 3:3, 23.

KELLER K.C., SAPORTA S.
1957
The frequency of consonant clusters in Chontal. International Journal of American Linguistics, 23, 28-35.

KNOPF
1981
Word Play. The Spoken Word, Peter Pan, New York, 262-264.

LEIFER L.
1981
Rehabilitative robots. Robotics Age, May/June, 4-18.

LLOYD L.L., FULLER D.R.
1986
Toward an augmentative and alternative communication taxonomy: A proposed superordinate classification. Augmentative and Alternative Communication, 2:4, 165-171.

LLOYD L.L., KARLAN G.R.
1984
Nonspeech communication symbols and systems: Where have we been and where are we going? Journal of Mental Deficiency Research, 28, 3-20.

LOOMIS J., POIZNER H., BELLLUGI U., BLAKEMORE A., HOLLERBACH J.
1983
Computer graphic modeling of American Sign Language. Siggraph Conference.

LYONS J.
1970
Chomsky. London: Fontana.

MACDONALD E.
1980
Teaching and Using Blissymbolics. Toronto: Blissymbolics Communication Institute.

MALONE K.
1936
The phonemic structure of English monosyllables. American Speech, 11, 205-218.

MAYZNER M.S., TRESSELT M.E.
1965
Tables of single-letter and digram frequency counts for various word-length and letter-position combinations. Psychonomic Monograph Supplements, 1 (2), 13-32.

MAYZNER M.S., TRESSELT M.E., WOLIN B.R.
1965
Tables of trigram frequency counts for various word-length and letter-position combinations. Psychonomic Monograph Supplements, 1 (3), 33-78.

MAYZNER M.S., TRESSELT M.E., WOLIN B.R.
1965a
Tables of tetragram frequency counts for various word-length and letter-position combinations. Psychonomic Monograph Supplements, 1 (4), 79-142.

MAYZNER M.S., TRESSELT M.E., WOLIN B.R.
1965b
Tables of pentagram frequency counts for various word-length and letter-position combinations. Psychonomic Monograph Supplements, 1 (5), 145-185.

MCKEAN K.
1985
In search of the unconscious mind. Discover, February, 12-16.

MCNAUGHTON S.
1985
Communicating with. Blissymbolics. Toronto: Blissymbolics Communication Institute.

MCNAUGHTON S.
1985a
Blissymbolics Independent Study Guide. Toronto: Blissymbolics Communication Institute.

MEYERS L.
1983
Micro-computers - Links to language: the early years. Communication Outlook 5:2, 20.

MINNEMAN S.L.
1984
A telecommunications aid for the deaf: An algorithm for extracting readable English text from touch-tone keystroke sequences. Department of Mechanical Engineering, Tufts - New England Medical Center.

MINSKY M.
1985
Why people think computers can't. In: Donnelly D.P. (Ed.) The computer culture. London: Associated University Presses, 27-43.

NEWMAN E.B.
1951
The pattern of vowels and consonants in various languages. American Journal of Psychology, 64, 369-379.

NEWMAN E., GERSTMANN L.
1952
A new method for analyzing printed English. Journal of Experimental Psychology, 44, 114-125.

O'CONNOR J.D., TRIM J.L.M.
1953
Vowel, consonant, and syllable a phonological definition. Word, 9:2, 103-122.

PAPERT S.
1980
Mindstorms. Brighton, Sussex: Harvester Press Ltd.

PICKERING J.A., ARNOTT J.L., WOLFF J.G., SWIFFIN A.L.
1984
Prediction and adaption in a communication aid for the disabled. Proceedings of the IFIP Conference on Human-Computer Interaction, London, 169-173.

PILGRIM R.
1975
Employment. In Personal Computers and Special Needs, 32-36.

POON P.
1983
MACApple product information, Dept. Electronic and Electrical Eng., Kings College, Strand, London, UK.

RAWLINSON G.E.
1976
Bigram frequency counts and anagram lists. Quarterly Journal of Experimental Psychology, 28, 125-142.

RUBIN D.C.
1978
Word-initial and word-final ngram frequencies. Journal of Reading Behaviour, 10, 171-183.

SAPORTA S.
1955
Frequency of consonant clusters. Language, 31, 25-30.

SCHANK R.C.
1985
The problem of natural language. In: Donnelly D.P. (Ed.) The computer culture. London: Associated University Presses, 44-69.

SHANNON C.E.
1951
Prediction and entrophy of printed English. Bell System Technical Journal, 30, 50-64.

SIGUARD B.
1968
Rank-frequency distributions for phonemes. Phonetica, 18, 1-15.

SOEDE M., FOULDS R.A.
1986
Dilemma of prediction in communication aids and mental load. Proceedings of the Ninth Annual Conference on Rehabilitation Engineering, Minneapolis, Mn., 357-359.

SOLSO R.L.
1979
Positional frequency and versatility of letters for six-seven-, and eight-letter English words. Behaviour Research Methods and Instrumentation, 11, 355-358.

SOLSO R.L., JUEL C.L., RUBIN D.C.
1982
The frequency and versatility of initial and terminal letters in English words. Journal of Verbal Learning and Verbal Behaviour, 21, 220-235.

STEELE R.D.
1983
Stenotyped input in rehabilitation applications. Proceedings of the Six Annual Conference on Rehabilitation Engineering, San Diego, Ca., 168-170.

STOFFEL D.
1982
Talking terminals. Byte, 7:9, 276-314.

SUTTER E.
1983
An oculo-encephalographic communication system. Proceedings of the Sixth Annual Conference on Rehabilitation Engineering, San Diego, Ca., 242-244.

SWIFFEN A.L., ARNOTT J.L., NEWELL A.F.
1987
The use of syntax in a prediction communication aid for the physically handicapped. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 124-126.

TREVIRANUS J., NORRIS L.
1987
Predictive programs: writing tools for severely physically disabled students. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 130-132.

VANDERHEIDEN G.C.
1983
Non-conversational communication technology needs of individuals with handicaps. Rehabilitation World, Summer, 8-12

VANDERHEIDEN G.C.
1976
Providing a child with the means to indicate. In G.C. Vanderheiden, K. Grilley (Eds.), Non-vocal communication techniques and aids for the severely physically handicapped. Baltimore: University Park Press, 20-76.

VANDERHEIDEN G.C., LLOYD L.L.
1986
Communication systems and their components. In S.W. Blackstone (Ed.), Augmentative Communication: An Introduction. Rockville: American Speech and Hearing Association, 49-162.

WALLER A.
1987
Augmentative communication begins in South Africa. Communicating Together, 5:3, 7-8.

WANG W.S-Y., CRAWFORD J.
1960
Frequency studies of English consonants. Language and Speech, 3, 131-139.

WEIR S., RUSSELL S.J., VALENTE J.A.
1982
Logo: an approach to educating disabled children. Byte, 7:9, 342-360.

WINSTON P.H., HORN B.K.P.
1984
Lisp. Addison-Wesley.

WITTEN I.A., CLEARY J.G., DARRAGH J.J., HILL D.R.
1982
Reducing keystroke counts with a predictive communication interface. IEEE, 3, p.10.

YOUNG R.E.
1983
Mental handicap and electronics. Wireless Word, September, 24-29.

APPENDIX 1

GLOSSARY OF TERMS

Accessing - The method by which an individual physically interacts with a communication device.

Alternative communication - A different communication system, e.g. a typewriter.

Augmentative communication - An additional and/or complementary communication system, e.g. signing.

Activation - The physical closing of a switching device which acts as an input to an electronic device.

Ergonomic - Taking human performance factors into consideration - i.e. speed, accuracy, co-ordination - when designing and implementing a communication device.

Independent decision control (i.d.c.) - The number of mutually exclusive choices which can be made, e.g. the number of switches which can be activated.

Input switch - A physical device which, once activated, sends a signal to a computer.

Linguistic item - Any linguistic structure, e.g. letter, word, symbol, phrase, sentence.

Matrix - The arrangement of items in a row - column layout.

Messaging - Preparing short, informal messages and notes.

Openness - The degree of openness of a communication system refers to its versatility to communicate any message.

Parsing - The process by which a sentence is resolved into its component parts for subsequent analysis.

Pointing - The act of identifying an item to be selected.

Selection - The act of choosing an identified item.

Scanning - The selection of a 'unit' from a number of choices.

Search time - The time taken to identify cognitively, and visually locate an item.

Target - The desired item which needs to be selected.

APPENDIX 2

SOFTWARE DEVELOPMENT DETAILS

INTRODUCTION

An overall description of the software development can be found in chapter six. This development is discussed in more detail in this appendix. The data structures and program control are explained, as are specific functions. The programming code may be found in appendix 6.

PROGRAM DEVELOPMENT DETAILS

The overall structure of the software is presented in figure A2-1:

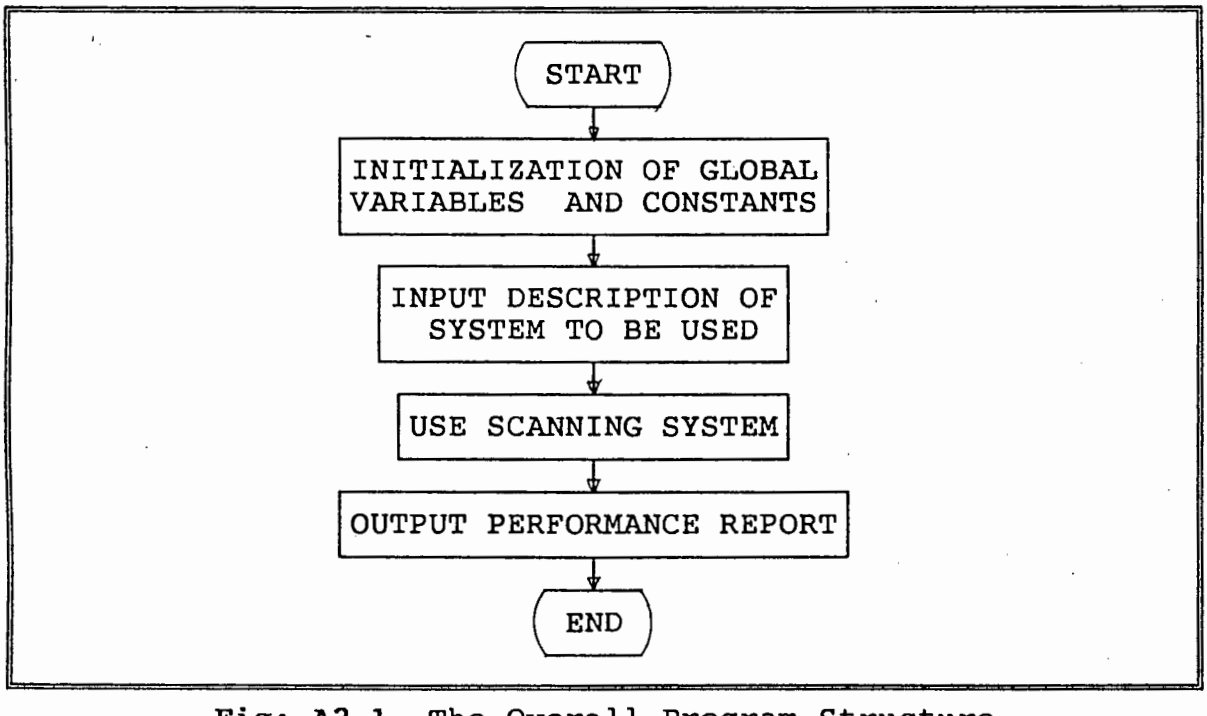


Fig: A2-1 The Overall Program Structure

The software is modular and was designed in a top-down, hierarchical fashion. Figure A2-2 illustrates this structure in terms of input, output and processing functions. The file names correspond to the actual files in which the various LISP functions are stored:

TYPE	NAME	DESCRIPTION OF PROCEDURES
INPUT FILES:	Read	- input.
	Readdig	- read in digrams.
	Rulebas	- input of rule-base.
OUTPUT FILES:	Cursor	- cursor movement.
	Screen	- monitor printing.
	Runout	- file storage for runs.
	Stats	- statistical reporting.
I/O FILES:	Fileio	- file handling.
PROCESSING FILES:	Access	- accessing environmental features and grouping attributes.
	Edit	- main processing.
	Init	- initialization factors.
	Loadup	- for booting system.
	Main	- Main program.
	Ruleman	- manipulation of rule-base.
	Scanner	- scanning system.
	Simulat	- scanning simulation.
	Util	- basic utilities.

Fig: A2-2 File structure containing procedure code.

DATA STRUCTURES

The list forms the basic structure of all data structures in LISP. Apart from the many simple variables and constants which are used in the program, a handful of more complicated data structures can be identified.

Stacks and Lists

Stacks are implemented using ordinary lists which are extended or reduced when necessary. The most important stacks are used to keep track of the group usage; and the text which is produced by the system.

The command which assigns the null list to a variable is:

(CLEAR-LIST <list-name>) => NIL

In order to add a new sublist to an existing list, the UPDATE-LIST command is used:

(UPDATE-LIST <list-name> <new-addition>) => list contents

The REMEMBER command adds a new sublist to an existing list only if the sublist does not already appear in the list:

(REMEMBER <list-name> <new-addition>) => list contents

Two commands are used to shorten lists thereby modifying their contents. They are:

(FORGET-FIRST <list-name>) => tail of list
(FORGET-LAST <list-name>) => beginning of list

Membership commands facilitate the interrogation of list variables. Two such commands are available:

(MEMBER= <list-to-be-matched> <list-name>) => true/nil
(INGROUP-P <item-to-be-matched> <list-name>) => true/nil

Window Streams

Windows are an essential part of the interactive nature of the software and makes it possible to control the outlay on the monitor. In order to create a window, the following attributes have to be specified: the height and width of the window; and xy-coordinates of the top left hand corner of the window (figure A2-3). Each grouping and the edit screen are provided with a window stream. A Golden Common LISP function, MAKE-WINDOW-STREAM is used to initialize such a window.

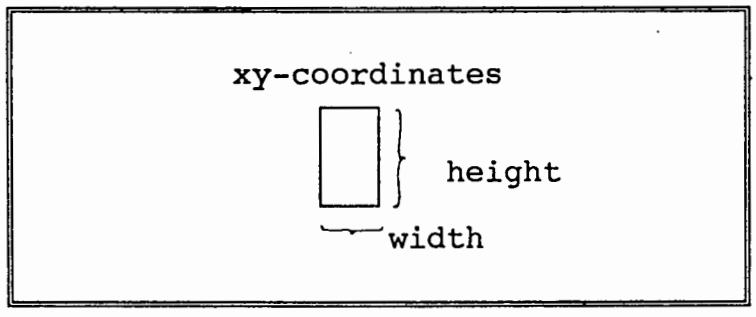


Fig: A2-3 Attributes required when creating a window stream.

Group Variables

During the initialization procedures, each grouping is bound to a symbol which then possesses various attributes. Each grouping has the following attributes: the name; the items comprising the group; the xy-coordinates of the top left hand corner of the group when on the screen; and the window stream associated with the grouping.

The creation and interrogation of these variables are essential for rule structure and were designed for easy use. The command, MAKE-GROUP-VARIABLE, creates a group variable

(fig. A2-4) and is used as follows:

(UPDATE-ENVIRONMENT <group-name> ({({column-item})})
<x-coordinate> <y-coordinate>) => name

The command GET-GROUP allows for interrogation of a group variable and is used as follows:

(GET-GROUP <group-name> <attribute>) => value

where <attribute> is one of the following:

:GROUP => group items
:STARTX => x-coordinate
:STARTY => y-coordinate
:COL => number of columns
:ROW => number of rows
:ENDX => x-coordinate
:ENDY => y-coordinate
:WINDOW => internal window stream
:ELEMENT <row> <col> => item in specified row and column

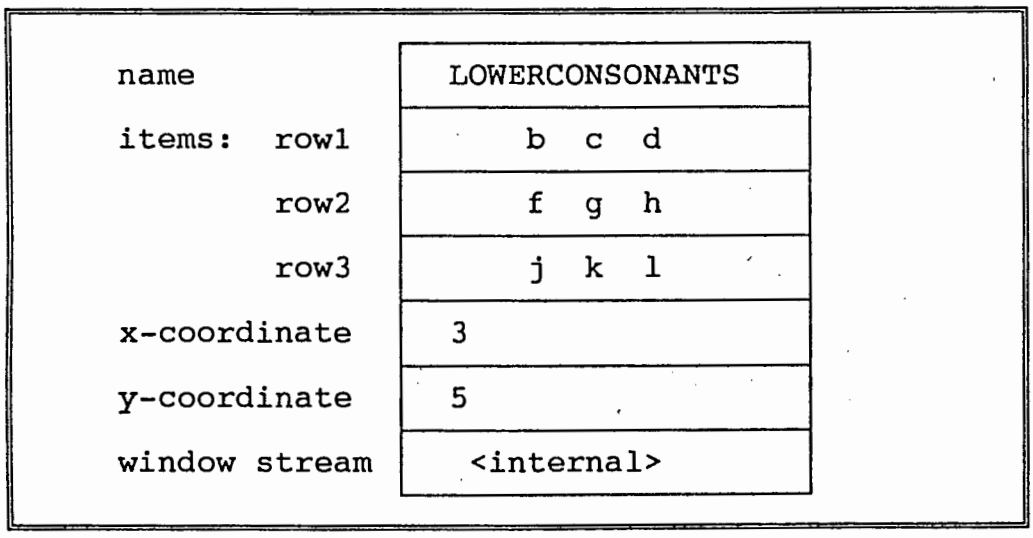


Fig: A2-4 Example of a group variable.

The environment also makes use of a symbol and corresponding attributes. Figure A2-5 illustrates the attributes associated with the current environment. The attributes which are stored are as follows: the grouping currently being scanned; the case (upper or lower) currently in use; the rule in use; the most recent item selected; the status of the environment (i.e. has the item been processed or not; and does the rule manipulator have to sift through the remaining rules or start from scratch); the position of the item within the linguistic frame; and an optional description attribute which is used for some rule matchings.

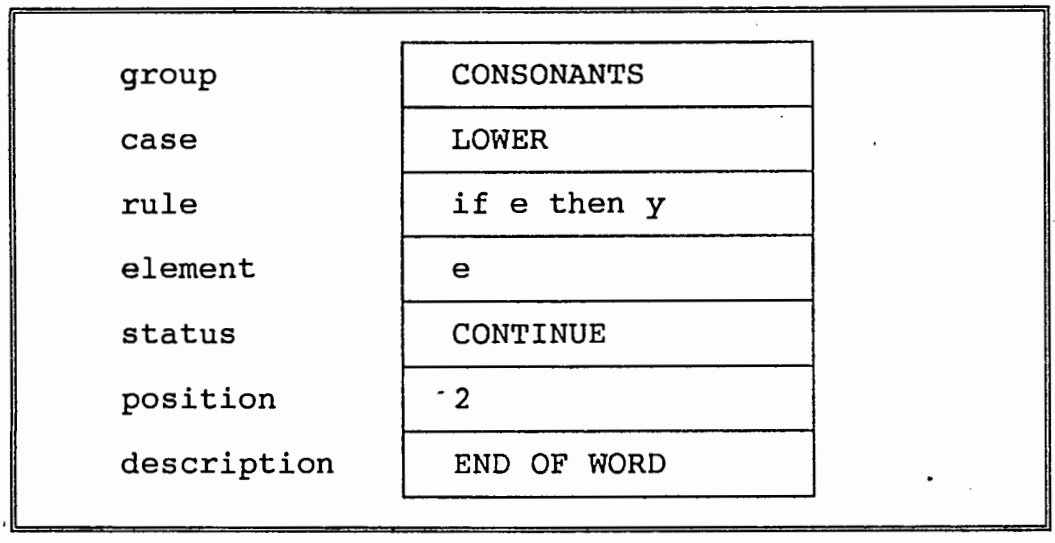


Fig: A2-5 Example of the environment attributes.

There are several functions which allows manipulation of the environment. The major function modifies the attributes and is accessed by issuing an UPDATE-ENVIRONMENT command. The syntax associated with this command is as follows:

(UPDATE-ENVIRONMENT {:<attribute> <value>}) => update result

where <attribute> can be any of: GROUP CASE
RULE ELEMENT
STATUS POSITION
DESCRIPTION

In order to interrogate the value of an environmental attribute, the command GET-ENVIRONMENT is used:

(GET-ENVIRONMENT :<attribute>}) => value

where <attribute> is as above.

The command NOTE-ENVIRONMENT returns the current values of all the environmental attributes and has no parameters:

(NOTE-ENVIRONMENT) => list containing attributes and values.

PROCEDURE IMPLEMENTATION

The software consists of a great number of procedures which determine tasks required by the system. Because of the nature of LISP, even the most fundamental tasks have to be specified, e.g. detection of an input from the keyboard requires the writing of programming code. This phenomenon is hardly a limitation as it allows the programmer the freedom to develop software which meets all system requirements.

As all the software is documented (appendix 6), only those procedures which determine scanning and rule manipulation will be highlighted.

Scanning

The scanning procedures have to be general enough to allow for any items to be scanned. The only specification is that a row/column scanning procedure must be applied to a list of items.

The procedure SCAN takes as its parameter the group to be scanned in the form of a list variable. The columns within the group are specified by sublists. It returns as its result a chosen item or the null list. Figure A2-6 illustrates the data flow between the procedures called by SCAN.

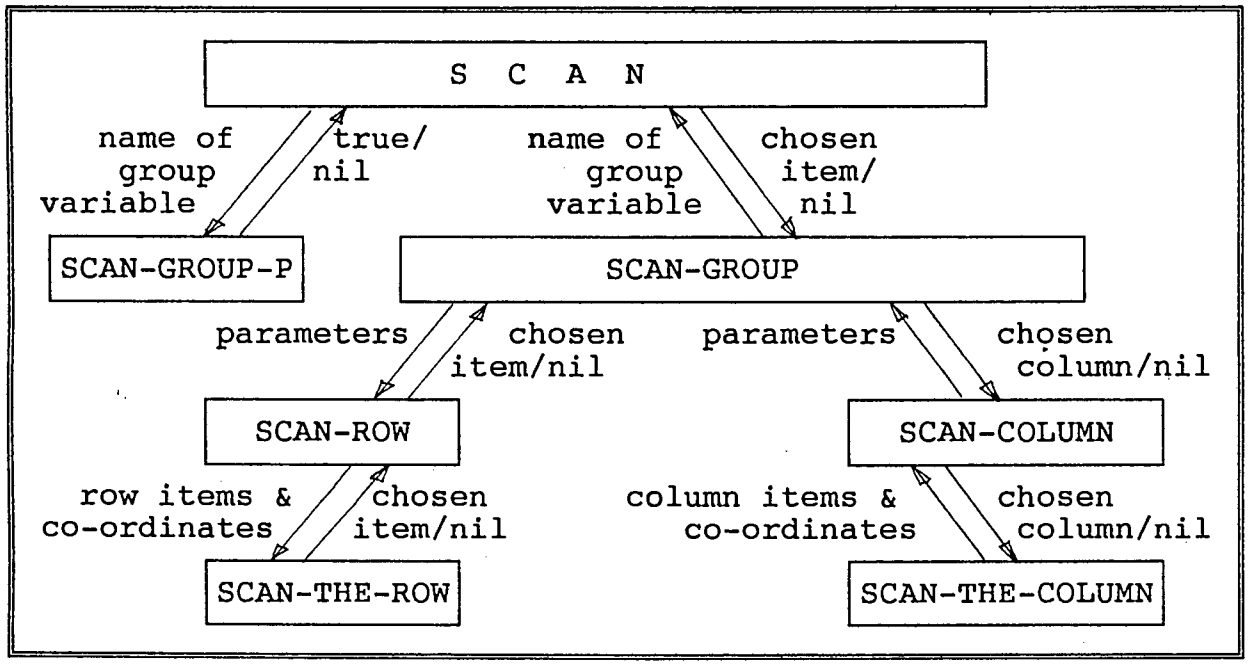


Fig: A2-6 Data flow of scanning procedures.

If more than one group has to be scanned, the group is highlighted first. If a switch interrupt is detected, the group is scanned, else a null response is returned.

When the group is scanned, each column is highlighted in turn until a switch interrupt is detected. The recursive procedure,

SCAN passes sublist by sublist to the SCAN-COLUMN procedure.

Once a switch interrupt has been detected, the relevant column is passed back to the procedure SCAN as a list. The SCAN-ROW procedure then highlights the individual items in the list and returns to SCAN only when another switch interrupt is detected. If no switch interrupt is initiated, the list is resubmitted to SCAN-ROW. However, selection of a dummy item, inserted by the procedure, allows for termination of the sequence.

In order to scan a specific grouping, a call to the function SCAN is required with the relevant grouping as parameter.

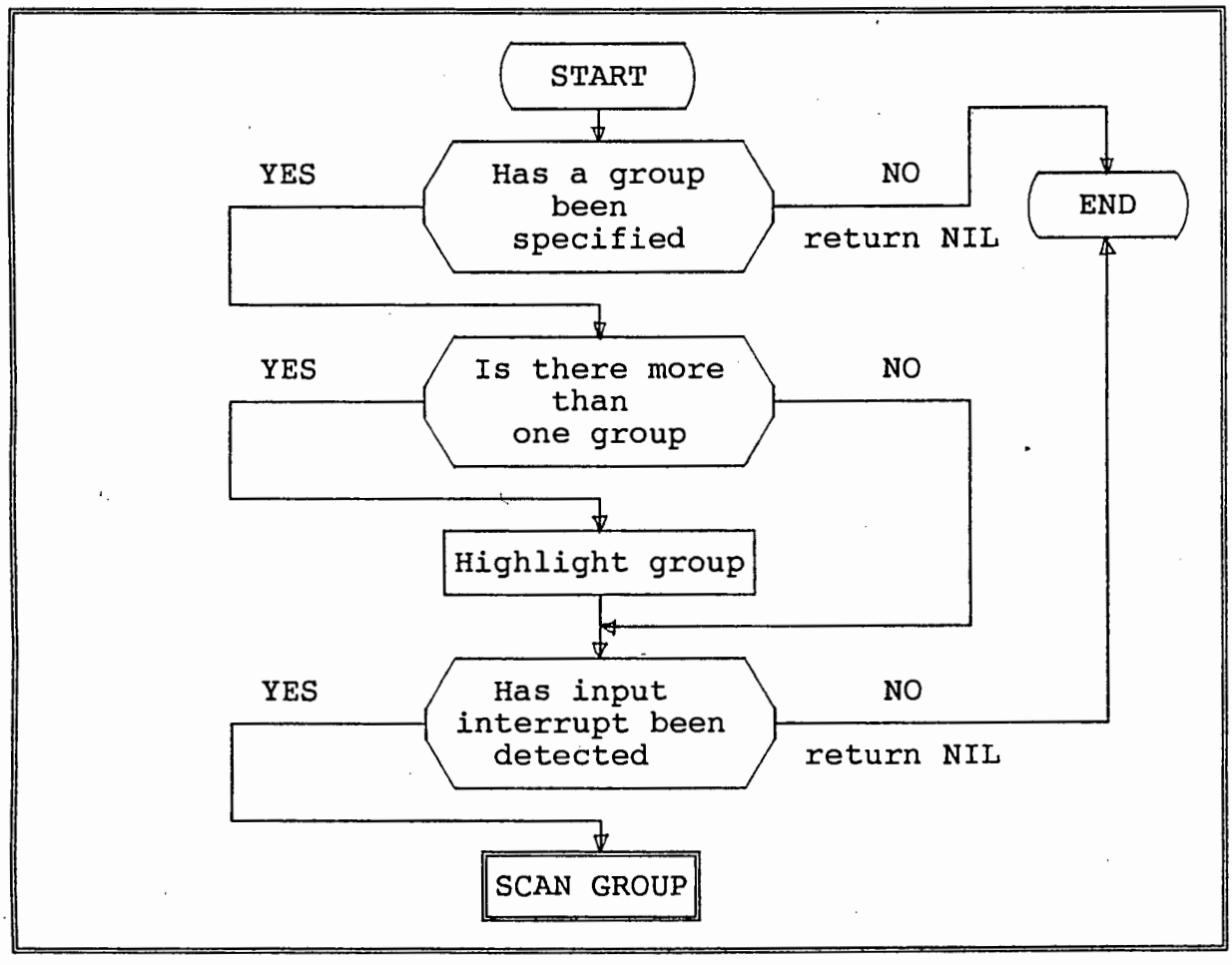


Fig:A2-7 (a) Flow chart of top level scanning procedure.

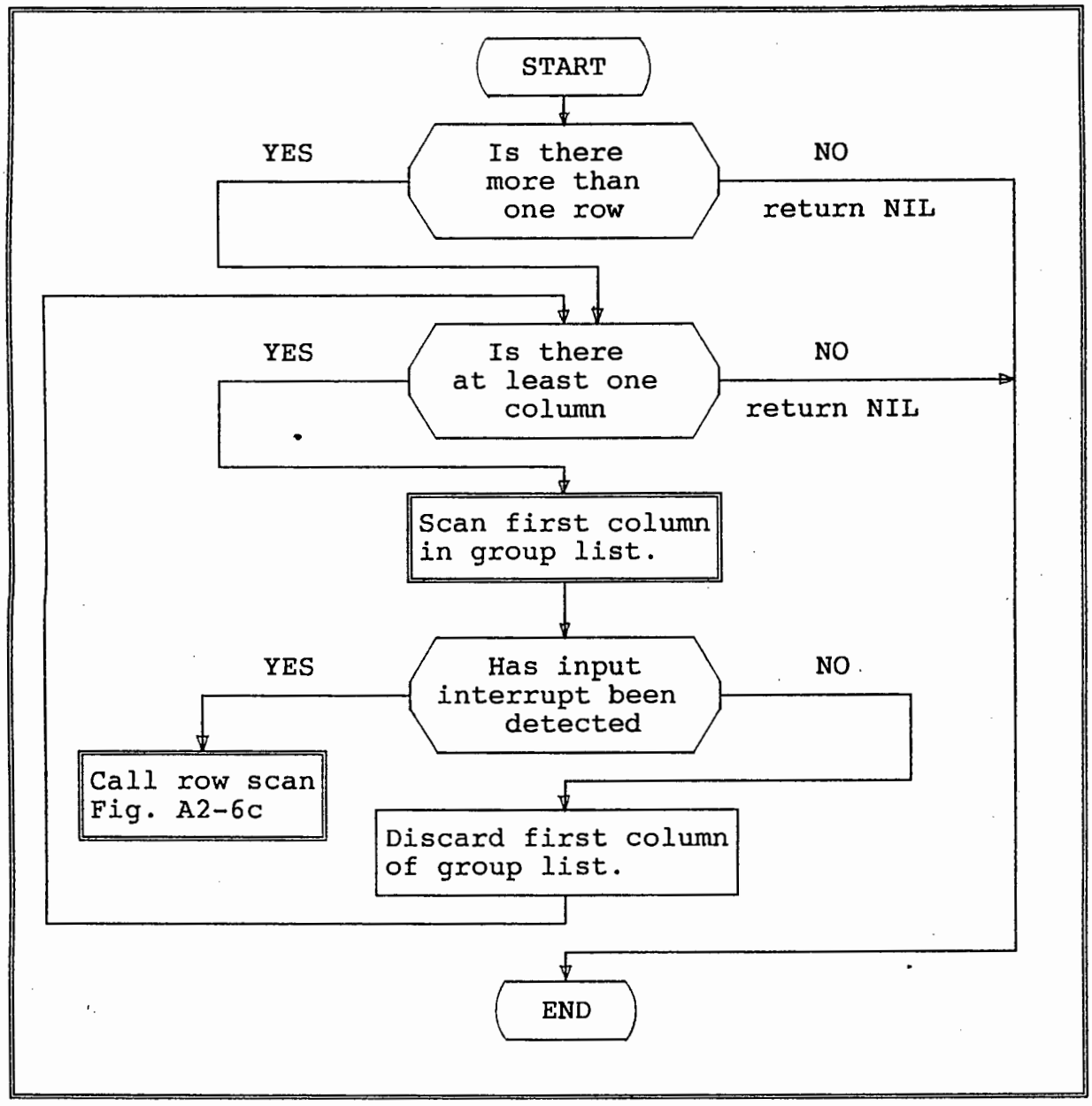


Fig: A2-7 (b) Flow chart of column scanning procedure.

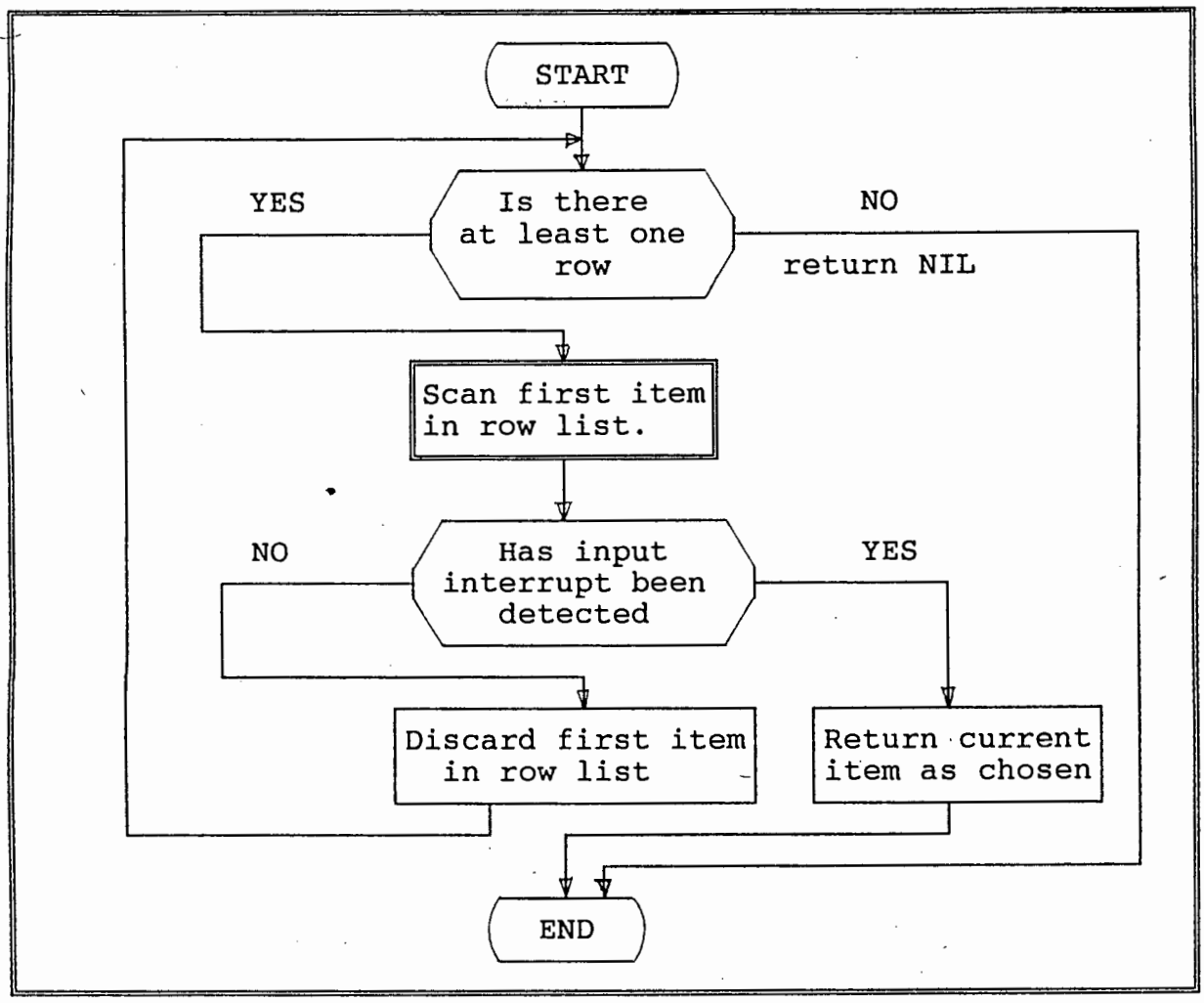


Fig: A2-7 (c) Flow chart of row scanning procedure.

Rule Manipulation

The variable, *RULE-BASE*, is a list structure which stores the rule base of a particular system being implemented. A rule of the system consists of an antecedent and a consequent. When setting up a description file, the rules required to describe the system are added to the rule base as follows:

(REMEMBER *RULE-BASE* (<antecedent> <consequent>))

The function which manipulates the rule base is called SIFT-RULES. This function tests each rule antecedent in turn until

one succeeds which results in the execution of the associated consequent. The rule manipulator then checks the environment to see whether or not to test the remaining rules, else it returns power to its calling function, PROCESS-ELEMENT. The data and control flow of these processes are shown schematically in figures A2-8 and A2-9 (following page) respectively.

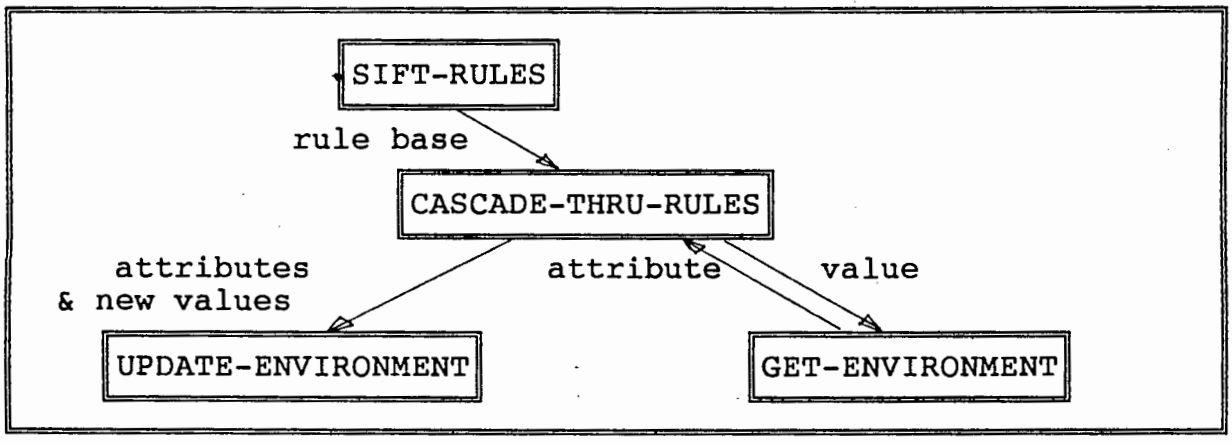


Fig: A2-8 Data flow of rule manipulation procedures.

Evaluation Procedure

Counters keep count of the following:

TOTAL-ELEMENTS	number of items chosen
KEYSTROKES-YES	number of switch activations
KEYSTROKES-NO	number of irrelevant scans
KEYSTROKES-DEL	number of deletions

This information is used to compile the necessary statistics:

NO SCAN : SCAN	ratio
EFFICIENCY	total scans / total elements
REDUNDANCY	(expected - observed) / expected

These statistics are stored in a file as are copies of the rule base, selected items and processed text.

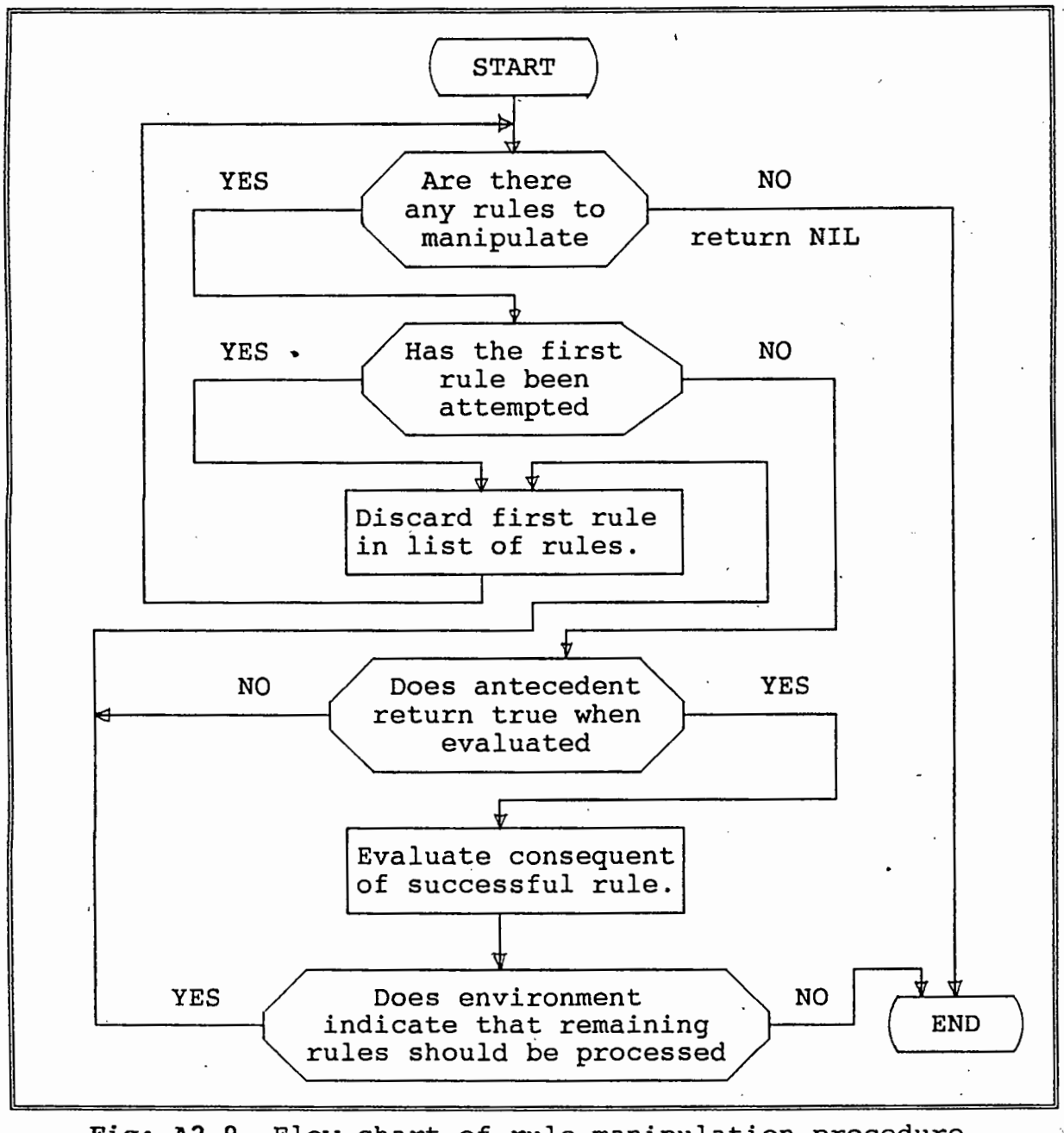


Fig: A2-9 Flow chart of rule manipulation procedure.

APPENDIX 3

RULE BASES FOR SCANNING SYSTEMS

"STATA" - STATIC STATISTICAL SYSTEM

```
;;=============================================================
;;FILE NAME:  STATIC
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with a static display of letters.
;;=============================================================

(DEFUN INITIALISE-GROUPS NIL

;;=============================================================
;;  Give a name to the technique being used
;;=============================================================

  (SETQ *TECHNIQUE* '"Static statistical")
  (SETQ *EFFICIENCY* 4.47)
  (SETQ *MODIFY-RULE* NIL)

;;=============================================================
;;  Indicate that only one group is displayed at any given time
;;=============================================================

  (SETQ *MULTIPLE-GROUPS* NIL)

;;=============================================================
;;  Set up variables
;;=============================================================

  (MAKE-GROUP-VARIABLE 'UPPERLETTERS
     `((,*INSERTSYM* "T" "O" "R" "U") ("E" "A" "N" "C" "P")
       (,*RETURNSYM* "I" "D" "Y" "Q") ("L" "M" "F" "X" "Z")
          ("S" "H" "G" "J" "W") ("B" "K" "V" ,*QUITSYM*)) 15 11)

;;=============================================================
;;  Read in rules
;;=============================================================

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
          (PRINT-CHOICE *BLANK*)))

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (UPDATE-ENVIRONMENT :CONTINUE)))

  (REMEMBER *RULE-BASE*
       ((MEMBER= (GET-ENVIRONMENT :ELEMENT) *QSET*)
          (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
          (PRINT-CHOICE *UPPU*)))
)
```

```
(DEFUN INITIATE-RULES NIL

;;=============================================================
;;  Set up intitial environment
;;=============================================================

  (UPDATE-ENVIRONMENT :GROUP 'LETTERS)
  (UPDATE-ENVIRONMENT :CASE  'UPPER)

;;=============================================================
;;  Position first cursor
;;=============================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*)
 )
```

"STATB" - CLARKE SYSTEM

```
;;=============================================================
;;FILE NAME:  STATIC
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with a static display of letters.
;;=============================================================

(DEFUN INITIALISE-GROUPS NIL

;;=============================================================
;;  Give a name to the technique being used
;;=============================================================

  (SETQ *TECHNIQUE* '"Static statistical")
  (SETQ *MODIFY-RULE* NIL)

;;=============================================================
;;  Indicate that only one group is displayed at any given time
;;=============================================================

  (SETQ *MULTIPLE-GROUPS* NIL)

;;=============================================================
;;  Set up variables
;;=============================================================

  (MAKE-GROUP-VARIABLE 'UPPERLETTERS
      (("E" ,*INSERTSYM* ,*RETURNSYM*) ("I" "M" "Y")
            ("A" "N" "D" "L") ("O" "U" "P" "K")
            ("R" "G" "C" "H") ("S" "B" "F" "J")
            ("T" "V" "W" "X") ("H" "G" "Z" "Q")) 15 11)

;;=============================================================
;;  Read in rules
;;=============================================================

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
          (PRINT-CHOICE *BLANK*)))
```

```
  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (UPDATE-ENVIRONMENT :CONTINUE)))

  (REMEMBER *RULE-BASE*
       ((MEMBER= (GET-ENVIRONMENT :ELEMENT) *QSET*)
          (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
          (PRINT-CHOICE *UPPU*)))

 )

(DEFUN INITIATE-RULES NIL

;;=================================================================
;;  Set up intitial environment
;;=================================================================

  (UPDATE-ENVIRONMENT :GROUP 'LETTERS)
  (UPDATE-ENVIRONMENT :CASE  'UPPER)

;;=================================================================
;;  Position first cursor
;;=================================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*)

 )
```

"DIGRAM" - DIGRAM SYSTEM

```
;;=============================================================
;;FILE NAME:  DIGRAM
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with digram letter anticipation.
;;=============================================================


(DEFUN INITIALISE-GROUPS NIL

;;=============================================================
;;  Give a name to the technique being used
;;=============================================================

  (SETQ *TECHNIQUE* '"Dynamic digrams")
  (SETQ *EFFICIENCY* 3.6)
  (SETQ *TOTAL-ELEMENTS* -1)

;;=============================================================
;;  Indicate that only one group is displayed at any given time
;;=============================================================

  (SETQ *MULTIPLE-GROUPS* NIL)

;;=============================================================
;;  Read in the name of the file containing the digrams
;;=============================================================

  (COND ((NULL *DIGRAM-NAME*) (READ-IN-DIGRAM-NAME)) (T ))

;;=============================================================
;;  Define a function to determine the digram groups.
;;=============================================================

  (DEFUN CREATE-DIGRAM-GROUP (NEWLIST)
    (COND (NEWLIST (CLEAR-LIST *DIGRAM-LIST*))
          (T))
    (CLEAR-SCREEN (GET-GROUP 'UPPERDIGRAM :WINDOW))
    (MAKE-GROUP-VARIABLE 'UPPERDIGRAM
      (LIST (APPEND (FIRST-BIT-DIGRAM  5)
                    (LIST *ASTERISK* *RETURNSYM* *QUITSYM*)))
       5 18)
    (UPDATE-ENVIRONMENT :GROUP 'DIGRAM))

;;=============================================================
;;  Set up rules
;;=============================================================

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *ASTERISK*)
           (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)
           (CREATE-DIGRAM-GROUP NIL)))
```

```
  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
          (PRINT-CHOICE *BLANK*)
          (FORGET-FIRST *NEWTEXT*)
          (SETF *NEWTEXT* (APPEND *NEWTEXT* (LIST *BLANK*)))
          (CREATE-DIGRAM-GROUP T)))

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (FORGET-FIRST *NEWTEXT*)
          (SETF *NEWTEXT* (APPEND *NEWTEXT* (LIST *BLANK*)))
          (CREATE-DIGRAM-GROUP T)))

  (REMEMBER *RULE-BASE*
       ((MEMBER= (GET-ENVIRONMENT :ELEMENT) *QSET*)
          (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
          (PRINT-CHOICE *UPPU*)
          (FORGET-FIRST *NEWTEXT*)
          (SETF *NEWTEXT* (APPEND *NEWTEXT* (LIST *UPPU*)))
          (CREATE-DIGRAM-GROUP T)))

  (REMEMBER *RULE-BASE*
      (T (FORGET-FIRST *NEWTEXT*)
         (SETF *NEWTEXT* (APPEND *NEWTEXT*
                                 (LIST (GET-ENVIRONMENT :ELEMENT))))
         (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
         (MATCH-A-DIGRAM (FIRST *NEWTEXT*)
                                 (CAR (LAST *NEWTEXT*)))
         (CREATE-DIGRAM-GROUP T)))

;;=============================================================
;;  Set up simulation rules
;;=============================================================

 (REMEMBER *SIMULATION-RULES*
      ((NOT (INGROUP-P ITEM (GET-GROUP 'UPPERDIGRAM :GROUP)))
             (LIST *ASTERISK* ITEM)))        ; Select next group
 )

(DEFUN INITIATE-RULES NIL
;;=============================================================
;;  Set up intitial environment
;;=============================================================

  (SETQ *NEWTEXT* (LIST *BLANK* *BLANK*))
  (CREATE-DIGRAM-GROUP T)
  (UPDATE-ENVIRONMENT :GROUP NIL
                      :CASE  'UPPER
                      :ELEMENT " ")

;;=============================================================
;;  Position first cursor
;;=============================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*) )
```

"LINGA" - LINGUISTIC - VERSION A

```
;;================================================================
;;FILE NAME:  DYNAMIC
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with a dynamic display of five different
;;  letters groups.
;;================================================================

(DEFUN INITIALISE-GROUPS NIL

;;================================================================
;;  Give a name to the technique being used
;;================================================================

  (SETQ *TECHNIQUE*         '"Dynamic linguistic - version A"
        *MULTIPLE-GROUPS* T)
  (SETQ *EFFICIENCY* 4.4)
  (SETQ *MAXEDITROW* 4)

;;================================================================
;;  Reset base scanning factor
;;================================================================

  (SETQ *BASE-DELAY* 3)

;;================================================================
;;  Set up variables
;;================================================================

  (SETQ *MAX-GROUPS* 5)

  (MAKE-GROUP-VARIABLE 'UPPERVOWELS
     `((,*INSERTSYM* "A" "I" ) ("E" "O" "U") ("S" ,*PUNÇTSYM*))
           60  10)
  (MAKE-GROUP-VARIABLE 'UPPERCONSONANTS
     `((,*INSERTSYM* "S" "C" "B" ) ("T" "F" "P" "J")
       ("D" "G" "K" ,*PUNCTSYM*))
            4 10)
  (MAKE-GROUP-VARIABLE 'UPPERSEMIS
     `((,*INSERTSYM* "R") ("N" "H") ("S" "M") ("L" "Y")
      ("V" ,*PUNCTSYM*))
           26  6)
  (MAKE-GROUP-VARIABLE 'UPPERFUNNIES
     `((,*INSERTSYM* "W") ("S" "Q") ("X" "Z") ("" ,*PUNCTSYM*))
           30 16)

  (MAKE-GROUP-VARIABLE 'UPPEREDIT
     `((,*DELETESYM* ,*RETURNSYM*) ("$+" "$-") (,*QUITSYM*))
           32 11)

;;  Set up hierarchical default groups
  (SETQ *HIERARCHY* '(CONSONANTS SEMIS VOWELS FUNNIES))
                                                    ;; RULE 11

;;  Set up rules
  (INITIALISE-RULES) )
```

```
(DEFUN INITIALISE-RULES NIL

;;=============================================================
;;  Set up rules
;;=============================================================

  (REMEMBER *RULE-BASE*
         ((EQUAL *MULTIPLE-GROUPS* NIL)
            (SETQ *MULTIPLE-GROUPS* T)
            (UPDATE-ENVIRONMENT :CONTINUE)))

   (REMEMBER *RULE-BASE*
        ((EQUAL (GET-ENVIRONMENT :ELEMENT)  *PUNCTSYM*)
         (RECORD-ENVIRONMENT)
         (UPDATE-ENVIRONMENT :STATUS      'PROCESSED
                             :GROUP       'EDIT)
         (SETQ *MULTIPLE-GROUPS* NIL)))

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (UPDATE-ENVIRONMENT :CONTINUE)))


  (REMEMBER *RULE-BASE*                                    ;; RULE 2
         ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
            (PRINT-CHOICE *BLANK*)
            (UPDATE-ENVIRONMENT :GROUP    NIL
                                :DESCRIPT NIL
                                :POSITION 0
                                :CONTINUE     )))

;; Set up hierarchy override

   (REMEMBER *RULE-BASE*                                   ;; RULE 12
        (T
          (UPDATE-ENVIRONMENT :GROUP
                     (FIND-POSSIBLE-GROUP *HIERARCHY*))))

;;=============================================================
;;  Set up simulation rules
;;=============================================================

 (REMEMBER *SIMULATION-RULES*
      ((INGROUP-P ITEM (GET-GROUP 'UPPEREDIT :GROUP))
                  (LIST *PUNCTSYM* ITEM)))   ; Select edit group

 )

(DEFUN INITIATE-RULES NIL

;;=============================================================
;;  Display all groups
;;=============================================================

  (MAPCAR 'DISPLAY-GROUP '(UPPERVOWELS UPPERCONSONANTS
                           UPPERFUNNIES UPPERSEMIS
                           UPPEREDIT))
```

```
;;=============================================================
;;   Set up intitial environment
;;=============================================================

  (UPDATE-ENVIRONMENT :GROUP 'CONSONANTS)
  (UPDATE-ENVIRONMENT :CASE  'UPPER)


;;=============================================================
;;  Position first cursor
;;=============================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*)

 )
```

"LINGB" - LINGUISTIC - VERSION B

```
;;================================================================
;;FILE NAME:  DYNAMIC
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with a dynamic display of five different
;;  letters groups.
;;================================================================

(DEFUN INITIALISE-GROUPS NIL

;;================================================================
;;  Give a name to the technique being used
;;================================================================

  (SETQ *TECHNIQUE*          '"Dynamic linguistic - version B"
        *MULTIPLE-GROUPS* T)
  (SETQ *EFFICIENCY* 4.4)
  (SETQ *MAXEDITROW* 4)

;;================================================================
;;  Reset base scanning factor
;;================================================================

  (SETQ *BASE-DELAY* 3)

;;================================================================
;;  Set up variables
;;================================================================

  (SETQ *MAX-GROUPS* 5)

  (MAKE-GROUP-VARIABLE 'UPPERVOWELS
     ((,*INSERTSYM* "A" "I" ) ("E" "O" "U") ("S" ,*PUNCTSYM*))
          60  10)
  (MAKE-GROUP-VARIABLE 'UPPERCONSONANTS
     ((,*INSERTSYM* "S" "C" "B" ) ("T" "F" "P" "J")
       ("D" "G" "K" ,*PUNCTSYM*))
            4 10)
```

```
  (MAKE-GROUP-VARIABLE 'UPPERSEMIS
    `((,*INSERTSYM* "R") ("N" "H") ("S" "M") ("L" "Y")
     ("V" ,*PUNCTSYM*))
          26  6)
  (MAKE-GROUP-VARIABLE 'UPPERFUNNIES
    `((,*INSERTSYM* "W") ("S" "Q") ("X" "Z") ("" ,*PUNCTSYM*))
          30 16)

  (MAKE-GROUP-VARIABLE 'UPPEREDIT
    `((,*DELETESYM* ,*RETURNSYM*) ("$+" "$-") (,*QUITSYM*))
          32 11)

;;  Set up hierarchical default groups
  (SETQ *HIERARCHY* '(CONSONANTS SEMIS VOWELS FUNNIES))
                                                        ;; RULE 11

;;  Set up rules
  (INITIALISE-RULES)

 )

(DEFUN INITIALISE-RULES NIL

;;=============================================================
;;  Set up rules
;;=============================================================

  (REMEMBER *RULE-BASE*
        ((EQUAL *MULTIPLE-GROUPS* NIL)
           (SETQ *MULTIPLE-GROUPS* T)
           (UPDATE-ENVIRONMENT :CONTINUE)))

   (REMEMBER *RULE-BASE*
        ((EQUAL (GET-ENVIRONMENT :ELEMENT)  *PUNCTSYM*)
         (RECORD-ENVIRONMENT)
         (UPDATE-ENVIRONMENT :STATUS     'PROCESSED
                             :GROUP      'EDIT)
         (SETQ *MULTIPLE-GROUPS* NIL)))

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (UPDATE-ENVIRONMENT :CONTINUE)))

   (REMEMBER *RULE-BASE*                                ;; RULE  3
        ((MEMBER= (GET-ENVIRONMENT :ELEMENT) *QSET*)
         (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
         (PRINT-CHOICE *UPPU*)
         (UPDATE-ENVIRONMENT :GROUP 'VOWEL
                             :DESCRIPT 'VOWEL-INITIAL
                             :POSITION 2)))
```

```
;; Start of linguistic theory

  (REMEMBER *RULE-BASE*                                          ;; RULE  2
        ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
           (PRINT-CHOICE *BLANK*)
           (UPDATE-ENVIRONMENT :GROUP    NIL
                               :DESCRIPT NIL
                               :POSITION 0
                               :CONTINUE     )))

   (REMEMBER *RULE-BASE*                                         ;; RULE 1
        ((AND (EQUAL (GET-ENVIRONMENT :NAME)    NIL)
              (EQUAL (GET-ENVIRONMENT :DESCRIPT) NIL)
              (EQUAL (GET-ENVIRONMENT :POSITION) 0))
         (UPDATE-ENVIRONMENT :GROUP    'CONSONANTS
                             :DESCRIPT 'WORD-INITIAL)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 4
        ((AND (EQUAL  (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL  (GET-ENVIRONMENT :POSITION) 0)
              (INGROUP-P (GET-ENVIRONMENT :ELEMENT) '"S"))))

   (REMEMBER *RULE-BASE*                                        ;; RULE 10
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-MEDIAL)
              (NOT (EQUAL (GET-ENVIRONMENT  :GROUP) 'VOWELS)))
          (UPDATE-ENVIRONMENT :GROUP 'CONSONANTS
                              :DESCRIPT 'WORD-INITIAL
                              :POSITION 0)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 5
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL (GET-ENVIRONMENT :POSITION) 0)
              (NOT (EQUAL    (GET-ENVIRONMENT :ELEMENT)  '"J")))
         (UPDATE-ENVIRONMENT :GROUP 'SEMIS
                             :POSITION 1)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 5
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL (GET-ENVIRONMENT :GROUP) 'CONSONANTS))
         (UPDATE-ENVIRONMENT :GROUP    'SEMIS)
                             :POSITION 1))

  (REMEMBER *RULE-BASE*                                         ;; RULE 7
        ((EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 0
                             :DESCRIPT 'VOWEL-INITIAL)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 7
        ((AND (EQUAL (GET-ENVIRONMENT :NAME) 'SEMIS)
              (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL))
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 0
                             :DESCRIPT 'VOWEL-INITIAL)))
```

```
   (REMEMBER *RULE-BASE*                                          ;; RULE 9
        ((INGROUP-P (GET-ENVIRONMENT :ELEMENT) '("W" "Q"))
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 1
                             :DESCRIPT 'VOWEL-INITIAL)))

   (REMEMBER *RULE-BASE*                                          ;; RULE 11
        ((EQUAL      (GET-ENVIRONMENT :NAME) 'VOWELS)
         (UPDATE-ENVIRONMENT :GROUP    'SEMIS
                             :POSITION 0
                             :DESCRIPT 'WORD-MEDIAL )))

;; Set up hierarchy override

   (REMEMBER *RULE-BASE*                                          ;; RULE 12
        (T
          (UPDATE-ENVIRONMENT :GROUP
                  .   (FIND-POSSIBLE-GROUP *HIERARCHY*))))

;;================================================================
;;  Set up simulation rules
;;================================================================

 (REMEMBER *SIMULATION-RULES*
      ((INGROUP-P ITEM (GET-GROUP 'UPPEREDIT :GROUP))
                  (LIST *PUNCTSYM* ITEM)))   ; Select edit group

 )

(DEFUN INITIATE-RULES NIL

;;================================================================
;;  Display all groups
;;================================================================

  (MAPCAR 'DISPLAY-GROUP '(UPPERVOWELS UPPERCONSONANTS
                           UPPERFUNNIES UPPERSEMIS
                           UPPEREDIT))

;;================================================================
;;   Set up intitial environment
;;================================================================

  (UPDATE-ENVIRONMENT :GROUP 'CONSONANTS)
  (UPDATE-ENVIRONMENT :CASE  'UPPER)


;;================================================================
;;  Position first cursor
;;================================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*)

 )
```

"LINGC" - LINGUISTIC - VERSION C

```
;;=============================================================
;;FILE NAME:  DYNAMIC
;;
;;  This file describes the environment creation and rule base
;;  for a wordprocessor with a dynamic display of five different
;;  letters groups.
;;=============================================================

(DEFUN INITIALISE-GROUPS NIL

;;=============================================================
;;  Give a name to the technique being used
;;=============================================================

  (SETQ *TECHNIQUE*         '"Dynamic linguistic - version C"
        *MULTIPLE-GROUPS* T)
  (SETQ *EFFICIENCY* 4.4)
  (SETQ *MAXEDITROW* 4)

;;=============================================================
;;  Reset base scanning factor
;;=============================================================

  (SETQ *BASE-DELAY* 3)

;;=============================================================
;;  Set up variables
;;=============================================================

  (SETQ *MAX-GROUPS* 5)

  (MAKE-GROUP-VARIABLE 'UPPERVOWELS
     `((,*INSERTSYM* "A" "I" ) ("E" "O" "U") ("S" ,*PUNCTSYM*))
           60  10)
  (MAKE-GROUP-VARIABLE 'UPPERCONSONANTS
     `((,*INSERTSYM* "S" "C" "B" ) ("T" "F" "P" "J")
       ("D" "G" "K" ,*PUNCTSYM*))
            4 10)
  (MAKE-GROUP-VARIABLE 'UPPERSEMIS
     `((,*INSERTSYM* "R") ("N" "H") ("S" "M") ("L" "Y")
      ("V" ,*PUNCTSYM*))
           26  6)
  (MAKE-GROUP-VARIABLE 'UPPERFUNNIES
     `((,*INSERTSYM* "W") ("S" "Q") ("X" "Z") ("" ,*PUNCTSYM*))
           30 16)

  (MAKE-GROUP-VARIABLE 'UPPEREDIT
     `((,*DELETESYM* ,*RETURNSYM*) ("$+" "$-") (,*QUITSYM*))
           32 11)

;;  Set up hierarchical default groups
  (SETQ *HIERARCHY* '(CONSONANTS SEMIS VOWELS FUNNIES))
                                                    ;; RULE 11
;;  Set up rules
  (INITIALISE-RULES)
 )
```

```
(DEFUN INITIALISE-RULES NIL
;;=============================================================
;;  Set up rules
;;=============================================================
  (REMEMBER *RULE-BASE*
        ((EQUAL *MULTIPLE-GROUPS* NIL)
           (SETQ *MULTIPLE-GROUPS* T)
           (UPDATE-ENVIRONMENT :CONTINUE)))

   (REMEMBER *RULE-BASE*
        ((EQUAL (GET-ENVIRONMENT :ELEMENT)  *PUNCTSYM*)
         (RECORD-ENVIRONMENT)
         (UPDATE-ENVIRONMENT :STATUS      'PROCESSED
                             :GROUP       'EDIT)
         (SETQ *MULTIPLE-GROUPS* NIL)))

  (REMEMBER *RULE-BASE*
       ((EQUAL (GET-ENVIRONMENT :ELEMENT) *RETURNSYM*)
          (NEW-LINE)
          (UPDATE-ENVIRONMENT :CONTINUE)))

   (REMEMBER *RULE-BASE*                                    ;; RULE  3
        ((MEMBER= (GET-ENVIRONMENT :ELEMENT) *QSET*)
         (PRINT-CHOICE (GET-ENVIRONMENT :ELEMENT))
         (PRINT-CHOICE *UPPU*)
         (UPDATE-ENVIRONMENT :GROUP 'VOWEL
                             :DESCRIPT 'VOWEL-INITIAL
                             :POSITION 2)))

;; Start of linguistic theory

  (REMEMBER *RULE-BASE*                                     ;; RULE  2
        ((EQUAL (GET-ENVIRONMENT :ELEMENT) *INSERTSYM*)
           (PRINT-CHOICE *BLANK*)
           (UPDATE-ENVIRONMENT :GROUP    NIL
                               :DESCRIPT NIL
                               :POSITION 0
                               :CONTINUE     )))

   (REMEMBER *RULE-BASE*                                    ;; RULE 1
        ((AND (EQUAL (GET-ENVIRONMENT :NAME)    NIL)
              (EQUAL (GET-ENVIRONMENT :DESCRIPT) NIL)
              (EQUAL (GET-ENVIRONMENT :POSITION) 0))
         (UPDATE-ENVIRONMENT :GROUP    'CONSONANTS
                             :DESCRIPT 'WORD-INITIAL)))

   (REMEMBER *RULE-BASE*                                    ;; RULE 4
        ((AND (EQUAL  (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL  (GET-ENVIRONMENT :POSITION) 0)
              (INGROUP-P (GET-ENVIRONMENT :ELEMENT) '"S"))))

   (REMEMBER *RULE-BASE*                                    ;; RULE 10
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-MEDIAL)
              (NOT (EQUAL (GET-ENVIRONMENT  :GROUP) 'VOWELS)))
          (UPDATE-ENVIRONMENT :GROUP 'CONSONANTS
                              :DESCRIPT 'WORD-INITIAL
                              :POSITION 0)))
```

```
   (REMEMBER *RULE-BASE*                                        ;; RULE 5
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL (GET-ENVIRONMENT :POSITION) 0)
              (NOT (EQUAL    (GET-ENVIRONMENT :ELEMENT)  '"J")))
         (UPDATE-ENVIRONMENT :GROUP 'SEMIS
                             :POSITION 1)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 5
        ((AND (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
              (EQUAL (GET-ENVIRONMENT :GROUP) 'CONSONANTS))
         (UPDATE-ENVIRONMENT :GROUP    'SEMIS)
                             :POSITION 1))

  (REMEMBER *RULE-BASE*                                         ;; RULE 7
        ((EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL)
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 0
                             :DESCRIPT 'VOWEL-INITIAL)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 7
        ((AND (EQUAL (GET-ENVIRONMENT :NAME) 'SEMIS)
              (EQUAL (GET-ENVIRONMENT :DESCRIPT) 'WORD-INITIAL))
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 0
                             :DESCRIPT 'VOWEL-INITIAL)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 9
        ((INGROUP-P (GET-ENVIRONMENT :ELEMENT) '("W" "Q"))
         (UPDATE-ENVIRONMENT :GROUP    'VOWELS
                             :POSITION 1
                             :DESCRIPT 'VOWEL-INITIAL)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 8
        ((AND (EQUAL (GET-ENVIRONMENT :NAME) 'VOWELS)
              (EQUAL (GET-ENVIRONMENT :POSITION) 0))
         (UPDATE-ENVIRONMENT :POSITION 1)))

   (REMEMBER *RULE-BASE*                                        ;; RULE 11
        ((AND (EQUAL (GET-ENVIRONMENT :NAME) 'VOWELS)
              (EQUAL (GET-ENVIRONMENT :POSITION) 1))
         (UPDATE-ENVIRONMENT :GROUP    'SEMIS
                             :POSITION 0
                             :DESCRIPT 'WORD-MEDIAL )))

;; Set up hierarchy override

   (REMEMBER *RULE-BASE*                                        ;; RULE 12
        (T
          (UPDATE-ENVIRONMENT :GROUP
              (FIND-POSSIBLE-GROUP *HIERARCHY*))))

 )
```

```
(DEFUN INITIATE-RULES NIL

;;=============================================================
;;  Display all groups
;;=============================================================

  (MAPCAR 'DISPLAY-GROUP '(UPPERVOWELS UPPERCONSONANTS
                           UPPERFUNNIES UPPERSEMIS
                           UPPEREDIT))

;;=============================================================
;;    Set up intitial environment
;;=============================================================

  (UPDATE-ENVIRONMENT :GROUP 'CONSONANTS)
  (UPDATE-ENVIRONMENT :CASE  'UPPER)

;;=============================================================
;;  Position first cursor
;;=============================================================

  (POSITION-CURSOR 0 0 *EDIT-WINDOW*)

 )
```

APPENDIX 4

ENGLISH DIGRAM LISTS [1] [2]

DIGRAMS WHERE LEADING LETTER IS: <space>

bbbABCDEFGHIJKLMNOPQRSTUVWXYZ
bANbSRLTCBPGMDFUWIVHEXQOKYZAJ
bBEUYOARLIbHFBDJSCGKMNPQTVWXZ
bCOAHLERUIbYFPCNBMZSJDGKQTVWX
bDEIOARUbWYHFMCSTBDGJKLNPQVXZ
bEXNVALMSFITCYDQURbPGOBJHKWEZ
bFORIAEULbTBCDFGHJKMNPQSVWXYZ
bGOREIAULbHTMYNBCDFGJKPQSVWXZ
bHAEIOUYbGSRBCDFHJKLMNPQTVWXZ
bINSTbFMDLRIGVCAQOXBHPZKUWEJY
bJUOAEbINCRBDFGHJKLMPQSTVWXYZ
bKNIEAROUGYCHbLWSTBDFJKMPQVXZ
bLIAEOUYRTbBDHSLNPCFGJKMQVWXZ
bMAOEUIYRNbMCLGPSDBFHJKQTVWXZ
bNOEAUIbGDKYSBTCFHJLMNPQRVWXZ
bOFNRUTPBVWCLHbMIXSADEKOGYZJQ
bPRAEOLUHISNbYCPTVBDFGJKMQWXZ
bQUbABCDEFGHIJKLMNOPQRSTVWXYZ
bREAOIUHbSPYBDMRCFGJKLNQTVWXZ
bSETOUHAIPCYMLKWbNQSRBDFGJVXZ
bTHOREAIWUYSbZNPTVBCDFGJKLMQX
bUNSPTbLRMGHBCDZAEFIJKOQUVWXY
bVIEAObUCLRSYBDFGHJKMNPQTVWXZ
bWHIAEORbUCNTBDFGJKLMPQSVWXYZ
bXbUVXAEIYBCDFGHJKLMNOPQRSTWZ
bYOEIAUbRVBCDFGHJKLMNPQSTWXYZ
bZEOIAWHUbBCDFGJKLMNPQRSTVXYZ

1. After: Gibler, 1981.

2. "b" indicates <space>.

DIGRAMS WHERE LEADING LETTER IS: A

bSCPMTFRLDWBGAHNOIVYJKUQEZXbb
AACRMLTbABDEFGHIJKNOPQSUVWXYZ
ABLOISRBYAEDUbNHTWJCFGKMPQVXZ
ACTHECKIRYQUOAbLDSBFMPGJNVWXZ
ADbEIVYDMOASURLJHNFCQBGTPWKXZ
AEMLbSCRNGAFTDOBEHIJKPQUVWXYZ
AFTFERbNAOILGKSBCDHJMPQUVWXYZ
AGEAGRINOMUbHSLDFPBCJKQTVWXYZ
AHbMAESIOLRDUNBCFGHJKPQTVWXYZ
AINDRLTMSGbCAEFVBIZKUWHJOPQXY
AJOEPbABCDFGHIJKLMNQRSTUVWXYZ
AKEIbSNDAFOTYUHKMBCGJLPQRVWXZ
ALbLISTEOUMYWAFRKVCDGBPNZHJQX
AMEbPIOABSMUYNLDWRHQTCFGJKVXZ
ANDbTCYSIGANOEKUXPQFWZLVHJBMR
AORbISTHMPABCDEFGJKLNOQUVWXYZ
APPAESITHbOYLRUBKNCDFGJMQVWXZ
AQUbABCDEFGHIJKLMNOPQRSTVWXYZ
ARETbIDYSRMALGKCNOBPFJXVWUHZQ
ASbTESIKOUPHCYAMBNLQGVRWDFJXZ
ATbIETHUOARCSMLYNFKBPZGJVDQWX
AUSTGDLbNCRBMXFPAEJUVZHIKOQWY
AVEIOAYRDLUbBCFGHJKMNPQSTVWXZ
AWbANSYKFIMELRDOCHBGJPQTUVWXZ
AXIbEAOPWBCDFGHJKLMNQRSTUVXYZ
AYbSEIAMBLDORTGFHYCJKNPQUVWXZ
AZIEbAOYZSUBCDFGHJKLMNPQRTVWX

DIGRAMS WHERE LEADING LETTER IS: B

BbTAIOCHWRDPSFLMBENUGYJVKQXZb
BASCRNBTLDYbGMKHFUIPAZJEOQVWX
BBIEOLAUbYCBDFGHJKMNPQRSTVWXZ
BCHObLUEABCDFGIJKMNPQRSTVWXYZ
BDUOIEbABCDFGHJKLMNPQRSTVWXYZ
BEbECRTILFHAGSDNYVWKPQZBJMOUX
BFIOUbABCDEFGHJKLMNPQRSTVWXYZ
BGRbABCDEFGHIJKLMNOPQSTUVWXYZ
BHAIOUbEMRBCDFGHJKLNPQSTVWXYZ
BILTNGRDSEOACbBZKHVPQFIJMUWXY
BJEUObABCDFGHIJKLMNPQRSTVWXYZ
BKbABCDEFGHIJKLMNOPQRSTUVWXYZ
BLEIYOAUbCBDFGHJKLMNPQRSTVWXZ
BMIAEUbBCDFGHJKLMNOPQRSTVWXYZ
BNOIAEbBCDFGHJKLMNPQRSTUVWXYZ
BOUTDROYNVLWASMXIbEBGKJCFHPQZ
BPbABCDEFGHIJKLMNOPQRSTUVWXYZ
BQbABCDEFGHIJKLMNOPQRSTUVWXYZ
BRIAOEUYbBCDFGHJKLMNPQRSTVWXZ
BSETObCIUABDFGHJKLMNPQRSVWXYZ
BTAbOESLFIRUBCDGHJKMNPQTVWXYZ
BUTSRDILNYCKBFGMZbEAHJOPQUVWX
BVIEObABCDFGHJKLMNPQRSTUVWXYZ
BWEAbBCDFGHIJKLMNOPQRSTUVWXYZ
BXbABCDEFGHIJKLMNOPQRSTUVWXYZ
BYbRSELAZBCDFGHIJKMNOPQTUVWXY
BZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: C

CbATSIPOCMRFWBDELHVNGUJKQYZXb
CALNTURSMPDbBVCIKEYGFJHOZAQWX
CBIESbABCDFGHJKLMNOPQRTUVWXYZ
CCEOUAILbRHBCDFGJKMNPQSTVWXYZ
CDObABCDEFGHIJKLMNPQRSTUVWXYZ
CEbSNRDPLIEMAFTCBYOKUZGHJQVWX
CFbAEORBCDFGHIJKLMNPQSTUVWXYZ
CGObABCDEFGHIJKLMNPQRSTUVWXYZ
CHbAIOENRULMTYSWFBCDGHKJPQVXZ
CIAEPOSTNDLRFVZBUMGbCHIYJKQWX
CJAbBCDEFGHIJKLMNOPQRSTUVWXYZ
CKbEISLYNGABOHWTPFMRUCDJKQVXZ
CLEAUOIYbBCDFGHJKLMNPQRSTVWXZ
CMbEABCDFGHIJKLMNOPQRSTUVWXYZ
CNETUAbBCDFGHIJKLMNOPQRSVWXYZ
CONMURLVSGPCTAbDOWHBIFEXQKYJZ
CPIbHABCDEFGJKLMNOPQRSTUVWXYZ
CQUbABCDEFGHIJKLMNOPQRSTVWXYZ
CREIOAUYbBCDFGHJKLMNPQRSTVWXZ
CSbTOSABCDEFGHIJKLMNPQRUVWXYZ
CTIbESOURLAFNMVTBCDGHJKPQWXYZ
CULRSTMPBEIAONUDFCJbGHKQVWXYZ
CVEbABCDFGHIJKLMNOPQRSTUVWXYZ
CWbABCDEFGHIJKLMNOPQRSTUVWXYZ
CXXbABCDEFGHIJKLMNOPQRSTUVWYZ
CYbTCANPLMSBDREFGHIJKOQUVWXYZ
CZEbABCDFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: D

DbTAIBOSWHFCPMNDERLUGVYJKQZXb
DAYTRNbMLIHSVBPUGWCKDZEAFJOQX
DBOAEIRYUbBCDFGHJKLMNPQSTVWXZ
DCAbEHLXBCDFGIJKMNOPQRSTUVWYZ
DDEIHLRbSYAOUBCDFGJKMNPQTVWXZ
DERbNSDAVCFPLMTEGBQOXIWUHZJKY
DFUAIbORBCDEFGHJKLMNPQSTVWXYZ
DGEMIKALWRSYbBCDFGHJNOPQTUVXZ
DHAIEORbUYBCDFGHJKLMNPQSTVWXZ
DISNTFCVAEDRLOGMUbPBZKXIQWHJY
DJUOAEbBCDFGHIJKLMNPQRSTVWXYZ
DKEUbABCDFGHIJKLMNOPQRSTVWXYZ
DLYEIAObDBCFGHJKLMNPQRSTUVWXZ
DMIOEAKbZBCDFGHJLMNPQRSTUVWXY
DNETIAOUbBCDFGHJKLMNPQRSVWXYZ
DObMWNEUCOLGRIPZSXDTVFABHJKQY
DPObIHABCDEFGJKLMNPQRSTUVWXYZ
DQUbABCDEFGHIJKLMNOPQRSTVWXYZ
DREAIOUbYSDBCFGHJKLMNPQRTVWXZ
DSbHTOMCILEAPBDFGJKNQRSUVWXYZ
DTbHIESABCDFGJKLMNOPQRTUVWXYZ
DUCARSELTMbPNGKOBDIVXFHJQUWYZ
DVEAIObBCDFGHJKLMNPQRSTUVWXYZ
DWAEIORbBCDFGHJKLMNPQSTUVWXYZ
DXbABCDEFGHIKJLMNOPQRSTUVWXYZ
DYbSNIEKLAUBCDFGHJMOPQRTVWXYZ
DZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: E

EbTASOICWPFMBHDRELNGUVKYJQZXb
EARSTDLCNbVKUMPGFBEHWAIJOQXYZ
EBRTEUAYOLIBbSCDFGHJKMNPQVWXZ
ECTOIEAHURLKbCYSDBFGJMNPQVWXZ
EDbIUEOGSLARWYDNMBTFHCJKPQVXZ
EENbDMPTKLRSICZBAVFHWXEGJOQUY
EFOFEIUbTLSARYBCDGHJKMNPQVWXZ
EGAIREUOSYLbGMNHDBCFJKPQTVWXZ
EHAIEOYUbRTMBCDFGHJKLNPQSVWXZ
EIRNGTVSLZDbBFKMCOEAPHIJQUWXY
EJEUOAbBCDFGHIJKLMNPQRSTVWXYZ
EKbSIELORABCDFGHJKMNPQTUVWXYZ
ELLYIFEAbODSVPTUMCGKBHRWZQJNX
EMbEPASOBIYNMULTDQCFGHJKRVWXZ
ENTbCDESGIOALURNVJFMHYBWQZKPX
EOPRUNVLbIGCFTSMDKABEHJOQWXYZ
EPTEARObHLIUPSYFWBCDGJKMNQVXZ
EQUbABCDEFGHIJKLMNOPQRSTVWXYZ
ERbESIANYTVMCFRPLGOHBWUDKJXZQ
ESbSTEIPUCOHANMKQDYLGFBRVJWXZ
ETbHEITWASYURCNOLBFPMZDGJKQVX
EURMCKTDSbPVNIXAEGLFBOHJQUWYZ
EVEIOAYbUSRTZBCDFGHJKLMNPQVWX
EWbSAHEIOBLMNDPCRYFGJKQTUVWXZ
EXPTCAIEbHUOQRLNSYBDFGJKMVWXZ
EYbEOSRAINBTDMVFHLCGJKPQUWXYZ
EZEbUHIOVAYZBCDFGJKLMNPQRSTWX

DIGRAMS WHERE LEADING LETTER IS: F

FbTASCHIMPWOERBFDLNGVYJUKQZXb
FACIRTMLSVNUBDbYGKXFHWAEJOPQZ
FBIbABCDEFGHJKLMNOPQRSTUVWXYZ
FCObABCDEFGHIJKLMNPQRSTUVWXYZ
FDbABCDEFGHIJKLMNOPQRSTUVWXYZ
FERCbENALWSDTMUIVGBZFHJKOPQXY
FFEIbOALSURHNMTYBCDFGJKPQVWXZ
FGHbABCDEFGIJKLMNOPQRSTUVWXYZ
FHAbBCDEFGHIJKLMNOPQRSTUVWXYZ
FICNRELGTVSFXDBAMZbHIJKOPQUWY
FJUbABCDEFGHIJKLMNOPQRSTVWXYZ
FKAIbBCDEFGHJKLMNOPQRSTUVWXYZ
FLUOEIAYbBCDFGHJKLMNPQRSTVWXZ
FMAEbBCDFGHIJKLMNOPQRSTUVWXYZ
FNEbABCDFGHIJKLMNOPQRSTUVWXYZ
FORULOCSNEWAIMXbGFKTBDHJPQVYZ
FPEbABCDFGHIJKLMNOPQRSTUVWXYZ
FQbABCDEFGHIJKLMNOPQRSTUVWXYZ
FROEAIUBQbCDFGHJKLMNPRSTVWXYZ
FSbPEHOABCDFGIJKLMNQRSTUVWXYZ
FTEbHSYILAWTBCDFGJKMNOPQRUVXZ
FULNRSTGCEMbDKABFHIJOPQUVWXYZ
FVbABCDEFGHIJKLMNOPQRSTUVWXYZ
FWAbBCDEFGHIJKLMNOPQRSTUVWXYZ
FXbABCDEFGHIJKLMNOPQRSTUVWXYZ
FYbIABCDEFGHJKLMNOPQRSTUVWXYZ
FZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: G

GbTAOISWHFPMBCEDRULNGYVJKQZXb
GAINTRLVSGbMZUPYBDEACKWFHJOQX
GBUYbABCDEFGHIJKLMNOPQRSTVWXZ
GCbABCDEFGHIJKLMNOPQRSTUVWXYZ
GDOAEbBCDFGHIJKLMNPQRSTUVWXYZ
GEbNRSTDMLOAIEVWBGHCFJKPQUXYZ
GFUIbABCDEFGHJKLMNOPQRSTVWXYZ
GGERLIAbSYOBCDFGHJKMNPQTUVWXZ
GHTbELOABIWSNFGMCDHJKPQRUVXYZ
GINVCSORMTBALFDbEGUZHIJKPQWXY
GJUbABCDEFGHIJKLMNOPQRSTVWXYZ
GKIbABCDEFGHJKLMNOPQRSTUVWXYZ
GLEAYIOUbBCDFGHJKLMNPQRSTVWXZ
GMEAbUBCDFGHIJKLMNOPQRSTVWXYZ
GNIbEOASMTUYBCDFGHJKLNPQRVWXZ
GOOVDbITNERALSUWMGZPCBFHYJKQX
GPIbABCDEFGHJKLMNOPQRSTUVWXYZ
GQbABCDEFGHIJKLMNOPQRSTUVWXYZ
GREAOIYUbMBCDFGHJKLNPQRSTVWXZ
GSbTIABOWCDEFGHJKLMNPQRSUVXYZ
GTHObBIACDEFGJKLMNPQRSTUVWXYZ
GUEIARLMSNYTOPZbFHJBCDGKQUVWX
GVbABCDEFGHIJKLMNOPQRSTUVWXYZ
GWAObBCDEFGHIJKLMNPQRSTUVWXYZ
GXbABCDEFGHIJKLMNOPQRSTUVWXYZ
GYbPARMBCDEFGHIJKLNOQSTUVWXYZ
GZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: H

HbTAIOHCWSMPFDBERLGNYVUJKQZXb
HATVNSDRLPIMBGbEWUKYCOZFHAJQX
HBOIRAbBCDEFGHJKLMNPQSTUVWXYZ
HCOILRbABCDEFGHJKMNPQSTUVWXYZ
HDRAbOUBCDEFGHIJKLMNPQSTVWXYZ
HEbRNYSIMADLTOEPCGWQFVUBKXZHJ
HFUIbABCDEFGHJKLMNOPQRSTVWXYZ
HGbOABCDEFGHIJKLMNPQRSTUVWXYZ
HIObIABCDEFGHJKLMNPQRSTUVWXYZ
HJbABCDEFGHIJKLMNOPQRSTUVWXYZ
HKEbABCDFGHIJKLMNOPQRSTUVWXYZ
HLYEAOIbBWCDFGHJKLMNPQRSTUVXZ
HMEAIbSOUBCDFGHJKLMNPQRTVWXYZ
HNIbOAESNHUTBCDFGJKLMPQRVWXYZ
HOUbWSLROMDPNICTEVAFGBZKHJYQX
HPIAUbBCDEFGHJKLMNOPQRSTVWXYZ
HQUbABCDEFGHIJKLMNOPQRSTVWXYZ
HROEIAUbMBCDFGHJKLNPQRSTVWXYZ
HSbITSEOAHMBCDFGJKLNPQRUVWXYZ
HTbESILFYHNRMOABUCDGJKPQTVWXZ
HUSMNRGATDLIBEbCFKOXYHUJPQVWZ
HVAbBCDEFGHIJKLMNOPQRSTUVWXYZ
HWAHIERbBCDFGJKLMNOPQSTUVWXYZ
HXbABCDEFGHIJKLMNOPQRSTUVWXYZ
HYbSLPTDMAGNCIRBEFHJKOQUVWXYZ
HZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: I

IbAWHTSDCMFEIBRKLPGNOUJVQYXZb
IALNTbRBSMGCHPZDEIOVAFJKQUWXY
IBLIEURBObSACNFYDGHJKMPQTVWXZ
ICAbHEITUKSYLORCMBDFGJNPQVWXZ
IDEbUIADLNOSGHWYBCKPRTFJMQVXZ
IESNDTVWRFbLCUGHMZBKPQYAEIJOX
IFbIFETYUOLASBCDGHJKMNPQRVWXZ
IGHNIEAUbOGMRSLBTYCDFJKPQVWXZ
IHOIAbEMTBCDFGHJKLNPQRSUVWXYZ
IIbISOLNEFABCDGHJKMPQRTUVWXYZ
IJAIObBCDEFGHJKLMNPQRSTUVWXYZ
IKEIUbAHKSLOBCDFGJMNPQRTVWXYZ
ILLIEbDYOASUTMVKFWGBHCRJNPQXZ
IMEPbIASMUBOLFGYNRCDHJKQTVWXZ
INbGTDESCAIVFKUNOLHJQYMBPWZRX
IONURDLbSTGZMCPVEAXBWFHIJKOQY
IPLbTAESPIMUHRBYOCDFGJKNQVWXZ
IQUbABCDEFGHIJKLMNOPQRSTVWXYZ
IRbESIMCTDARLOYGPBVFKWUNHJQXZ
ISbTHEMISCAOPDLFKRUBNGZQYJVWX
ITbHIYSEATUOLCRMNZFPQVBDGJKWX
IUMSbANDRBCEFGHIJKLOPQTUVWXYZ
IVEIAObYRUBCDFGHJKLMNPQSTVWXZ
IWAbOBCDEFGHIJKLMNPQRSTUVWXYZ
IXbTEOIAPBCDFGHJKLMNQRSUVWXYZ
IYAbBCDEFGHIJKLMNOPQRSTUVWXYZ
IZEAIOZbUVCDSBFGHJKLMNPQRTWXY

DIGRAMS WHERE LEADING LETTER IS: J

JbSACBHREFGMNOPVWDIJKLQTUXYZb
JAPNMSCBLRIYDWbHUVKTAEFGJOQXZ
JBbABCDEFGHIJKLMNOPQRSTUVWXYZ
JCPbABCDEFGHIJKLMNOQRSTUVWXYZ
JDbABCDEFGHIJKLMNOPQRSTUVWXYZ
JECRSAWNHTbELDOBFJMUGIKPQVXYZ
JFbABCDEFGHIJKLMNOPQRSTUVWXYZ
JGbABCDEFGHIJKLMNOPQRSTUVWXYZ
JHPbABCDEFGHIJKLMNOQRSTUVWXYZ
JINbMBGSTACDEFHIJKLOPQRUVWXYZ
JJbABCDEFGHIJKLMNOPQRSTUVWXYZ
JKbABCDEFGHIJKLMNOPQRSTUVWXYZ
JLbABCDEFGHIJKLMNOPQRSTUVWXYZ
JMbABCDEFGHIJKLMNOPQRSTUVWXYZ
JNDAbBCEFGHIJKLMNOPQRSTUVWXYZ
JORHYIBSUENbLKCADGFJMOPQTVWXZ
JPAUbBCDEFGHIJKLMNOPQRSTVWXYZ
JQbABCDEFGHIJKLMNOPQRSTUVWXYZ
JRbABCDEFGHIJKLMNOPQRSTUVWXYZ
JSbABCDEFGHIJKLMNOPQRSTUVWXYZ
JTbABCDEFGHIJKLMNOPQRSTUVWXYZ
JUSDRNMLIXATBGVbCEFHJKOPQUWYZ
JVbABCDEFGHIJKLMNOPQRSTUVWXYZ
JWbABCDEFGHIJKLMNOPQRSTUVWXYZ
JXbABCDEFGHIJKLMNOPQRSTUVWXYZ
JYbABCDEFGHIJKLMNOPQRSTUVWXYZ
JZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: K

KbTAOIWHSFMBCDPEYULRNGVJKQXZb
KAENBTGRLSbYIDMWFHPACJKOQUVXZ
KBUOEIRbABCDFGHJKLMNPQSTVWXYZ
KCObABCDEFGHIJKLMNPQRSTUVWXYZ
KDObABCDEFGHIJKLMNPQRSTUVWXYZ
KEbDNRTSLEYPWOFMGIVUABCHJKQXZ
KFAUILObBCDEFGHJKMNPQRSTVWXYZ
KGRbMABCDEFGHIJKLNOPQSTUVWXYZ
KHAMOESYbRBCDFGHIJKLNPQTUVWXZ
KINLbSTPDCMREAOBFGHIJKQUVWXYZ
KJbABCDEFGHIJKLMNOPQRSTUVWXYZ
KKIAbEBCDFGHJKLMNOPQRSTUVWXYZ
KLEYIOAbBCDFGHJKLMNPQRSTUVWXZ
KMAEbBCDFGHIJKLMNOPQRSTUVWXYZ
KNOEIAbBCDFGHJKLMNPQRSTUVWXYZ
KONRbVCSBUFLMOTAHDEGIJKPQWXYZ
KPEILObABCDFGHJKMNPQRSTUVWXYZ
KQbABCDEFGHIJKLMNOPQRSTUVWXYZ
KRUIAEOYbBCDFGHJKLMNPQRSTVWXZ
KSbHOAGCEIMBDFJKLNPQRSTUVWXYZ
KTHAbIOBCDEFGJKLMNPQRSTUVWXYZ
KULRbANMPBDKOCEFGHIJQSTUVWXYZ
KVbABCDEFGHIJKLMNOPQRSTUVWXYZ
KWAEObBCDFGHIJKLMNPQRSTUVWXYZ
KXbABCDEFGHIJKLMNOPQRSTUVWXYZ
KYbOLANRSTUBCDEFGHIJKMPQVWXYZ
KZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: L

LbTASOBCIPFWHRMDELNGVUYKJQZXb
LATRNSCWIYBbMUDHPGVLXKZEOFQAJ
LBEbORYASUILBCDFGHJKMNPQTVWXZ
LCOUEHAIKbBCDFGJLMNPQRSTVWXYZ
LDbEIRSANOHLWBUPTMCDFGJKQVXYZ
LEbSADMTCGNRVFXUEYIPLWBOHJKQZ
LFbIATROUEPWFSCBDGHJKLMNQVXYZ
LGAEIROUbBCDFGHJKLMNPQSTVWXYZ
LHIOAEUbBCDFGHJKLMNPQRSTVWXYZ
LITNSCEGKFAMVZOBDPbLQHUXIJYRW
LJbABCDEFGHIJKLMNOPQRSTUVWXYZ
LKbSEIUAYHLNOBCDFGJKMPQRTVWXZ
LLbYEIOASUNMPWFRVDLTBCGHJKQXZ
LMObESINAHLMBCDFGJKPQRTUVWXYZ
LNEUSAObBCDFGHIJKLMNPQRTVWXYZ
LOWNSGORPCYUVTADMBIbFQKHLZEJX
LPbEIFSHATLORBCDGJKMNPQUVWXYZ
LQUbABCDEFGHIJKLMNOPQRSTVWXYZ
LREbOYIASBCDFGHJKLMNPQRTUVWXZ
LSbOEIATHSDCPBFGJKLMNQRUVWXYZ
LTbHIESYOUARDZLBGTWCFJKMNPQVX
LUETSDMNRCAIBXGOPFLKbHJQUVWYZ
LVEIAUObBCDFGHJKLMNPQRSTVWXYZ
LWAOEISbBCDFGHJKLMNPQRTUVWXYZ
LXbABCDEFGHIJKLMNOPQRSTUVWXYZ
LYbSMITZRCDNOLFVWAGBEHJKPQUXY
LZLBbEUACDFGHIJKMNOPQRSTVWXYZ

DIGRAMS WHERE LEADING LETTER IS: M

MbTAOIWHSBFCMPRDENLGUYVJKQZXb
MANTYRLKIDSGbJCXMBPHZOUAEQFWV
MBEILAORbSUFNHYBCDGJKMPQTVWXZ
MCIDCNKGFHQVbABEJLMOPRSTUWXYZ
MDbCEOABDFGHIJKLMNPQRSTUVWXYZ
MEbNRDASTMLECWYOGHBXFKIPUVZJQ
MFQUEIbABCDFGHJKLMNOPRSTVWXYZ
MGbMSIRABCDEFGHJKLNOPQTUVWXYZ
MHAObBCDEFGHIJKLMNPQRSTUVWXYZ
MINLTSGCDAERXZbVOQUFMIYBHJKPW
MJbABCDEFGHIJKLMNOPQRSTUVWXYZ
MKIbABCDEFGHJKLMNOPQRSTUVWXYZ
MLYbEAILBCDFGHJKMNOPQRSTUVWXZ
MMOEUIAbYBCDFGHJKLMNPQRSTVWXZ
MNEbRAISLBCDFGHJKMNOPQTUVWXYZ
MORNSTVUDMCOLBKbGPZIAYEWHJFQX
MPLOAEHTRbUISBKYFNCDGJMPQVWXZ
MQUbABCDEFGHIJKLMNOPQRSTVWXYZ
MRbSAOITBCDEFGHJKLMNPQRUVWXYZ
MSbETOIFYHPSUACDKMRBGJLNQVWXZ
MTHAEbNBCDFGIJKLMOPQRSTUVWXYZ
MUSCLNMTREDFGHZIbABJKOPQUVWXY
MVEObABCDFGHIJKLMNPQRSTUVWXYZ
MWOEbABCDFGHIJKLMNPQRSTUVWXYZ
MXbABCDEFGHIJKLMNOPQRSTUVWXYZ
MYbSETOKCRAGBDFHIJLMNPQUVWXYZ
MZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: N

NbTAOIWSHBCFPMERDLNGUYVJKQZXb
NATLRbNPMGBDSCIEVKUWFYZHAJOQX
NBEOURAICLbBDFGHJKMNPQSTVWXYZ
NCEITHRLOYAUbKCSBDFGJMNPQVWXZ
NDbEISUAORLHMFYNPWTKBCDGJQVXZ
NEbSDRWCVETUAILNXYGMFOPZQBHJK
NFLOEIUARbBCDFGHJKMNPQSTVWXYZ
NGbESLIUTRADFHNOYMWBJCGKPQVXZ
NHAEIbOUSYBCDFGHJKLMNPQRTVWXZ
NINSTCOFZMEVGAQbUXPLDHRKBIJWY
NJUOAEIbBCDFGHJKLMNPQRSTVWXYZ
NKbSIERNLAYFHWBMOCDGJKPQTUVXZ
NLYIEAOLTbBCDFGHJKMNPQRSUVWXZ
NMEAIObBCDFGHJKLMNPQRSTUVWXYZ
NNEOIAbYUSHTBCDFGJKLMNPQRVWXZ
NOTWbRUMNSVPLBCDIOFXYGZEAHJKQ
NPRLOUEAHISTbBCDFGJKMNPQVWXYZ
NQUbABCDEFGHIJKLMNOPQRSTVWXYZ
NRYEbIUSAOBCDFGHJKLMNPQRTVWXZ
NSbTIECUAWPOFHMLKGYSVBRDNJQXZ
NTbEISRAOLHUYMGNFWDBCJKPQTVXZ
NUMETACIRSFLONbUGDPBHJKQVWXYZ
NVEOIAYUbBCDFGHJKLMNPQRSTVWXZ
NWAIOHERbBCDFGJKLMNPQSTUVWXYZ
NXIbABCDEFGHJKLMNOPQRSTUVWXYZ
NYbTOWSBMIALHDUCEFGJKNPQRVXYZ
NZbEOYAIBCDFGHJKLMNPQRSTUVWXZ

DIGRAMS WHERE LEADING LETTER IS: O

ObTABSHCMDIPWORFELNGUKYVJQZXb
OADCRTLNKSMPbXBFGHIUVAEJOQWYZ
OBLSJAETVbIBYOURWCDFNGHKMPQXZ
OCIEKCATYHRUOLbQBDFGJMNPSVWXZ
ODbUEYSIAGODNHMLBRWFPCJKQTVXZ
OESbTRLVDMANXCYFUBEHIKPGJOQWZ
OFbFTIEOAUSJLMBCDGHKNPQRVWXYZ
OGIREYNAbSUGLMOWTFBCDHJKPQVXZ
OHNbAEOIHLMBCDFGJKPQRSTUVWXYZ
OINCDSLTREBKAbXGFHIJMOPQUVWYZ
OJEOAbHBCDFGIJKLMNPQRSTUVWXYZ
OKbEISYALOKWBCDFGHJMNPQRTUVXZ
OLLDIOEbUVSYAKTFMBCNGPWZHRJQX
OMbEPMIABOSYFRNLKTUVCDGHJQWXZ
ONbSETGADCLFOIVNYKMQUWPJHRZBX
OODKLbRTNMPSFBVEHIOACGJQUWXYZ
OPELHPObMIUSTRAYKDFBCGJNQVWXZ
OQUbABCDEFGHIJKLMNOPQRSTVWXYZ
ORbETDMSKIYAGLCRNPOWBUFHVZJQX
OSETSIOPbACUMHYLQBKDFGJNRVWXZ
OTbHEITAOSLYCUBRMWNPZDFGJKQVX
OURLTNSGbPBCDVFIEKMQAHJOUWXYZ
OVEIAObLSBCDFGHJKMNPQRTUVWXYZ
OWbENSILATDYMHFCRBPWGJKOQUVXZ
OXIbYEFALBCDGHJKMNOPQRSTUVWXZ
OYbEASMCIOFHNDLPBGJKQRTUVWXYZ
OZEbAZIBCDFGHJKLMNOPQRSTUVWXY

DIGRAMS WHERE LEADING LETTER IS: P

PbTAOIWSHFBMCLPRDEUYNGJVKQXZb
PARTNSICYLPGBWUZKDbVEMHOXAFJQ
PBEUORbABCDFGHIJKLMNPQSTVWXYZ
PCSbCABDEFGHIJKLMNOPQRTUVWXYZ
PDAYbBCDEFGHIJKLMNOPQRSTUVWXZ
PERCNAObDTLSEPIKUXFGMYBJVWHQZ
PFUILObABCDEFGHJKMNPQRSTVWXYZ
PGRbABCDEFGHIJKLMNOPQSTUVWXYZ
PHIYEAORbSNULTBCDFGHJKMPQVWXZ
PINTRCELDPSAbGOKUFVBHIJMQWXYZ
PJbABCDEFGHIJKLMNOPQRSTUVWXYZ
PKEIbABCDFGHJKLMNOPQRSTUVWXYZ
PLEAIOYUbBCDFGHJKLMNPQRSTVWXZ
PMEbOABCDFGHIJKLMNPQRSTUVWXYZ
PNOEbABCDFGHIJKLMNPQRSTUVWXYZ
POSRNLIWTPOKUECVYMbDBGAXFHJQZ
PPEOLRIAYbUSBCDFGHJKMNPQTVWXZ
PQUbABCDEFGHIJKLMNOPQRSTVWXYZ
PROEIAUYbDFLBCGHJKMNPQRSTVWXZ
PSbYEIHTAOUCSBDFGJKLMNPQRVWXZ
PTbIEUSAOCHYLRBDFGJKMNPQTVWXZ
PURLBTNMSPZDGICEFYbAHJKOQUVWX
PVTbABCDEFGHIJKLMNOPQRSUVWXYZ
PWAbBCDEFGHIJKLMNOPQRSTUVWXYZ
PXbABCDEFGHIJKLMNOPQRSTUVWXYZ
PYbIOAEKLRSWTBCDFGHJMNPQUVXYZ
PZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: Q

QbAFMWCDHOSTBEGIJKLNPQRUVXYZb
QAbABCDEFGHIJKLMNOPQRSTUVWXYZ
QBbABCDEFGHIJKLMNOPQRSTUVWXYZ
QCbABCDEFGHIJKLMNOPQRSTUVWXYZ
QDbABCDEFGHIJKLMNOPQRSTUVWXYZ
QEbABCDEFGHIJKLMNOPQRSTUVWXYZ
QFbABCDEFGHIJKLMNOPQRSTUVWXYZ
QGbABCDEFGHIJKLMNOPQRSTUVWXYZ
QHbABCDEFGHIJKLMNOPQRSTUVWXYZ
QIbABCDEFGHIJKLMNOPQRSTUVWXYZ
QJbABCDEFGHIJKLMNOPQRSTUVWXYZ
QKbABCDEFGHIJKLMNOPQRSTUVWXYZ
QLbABCDEFGHIJKLMNOPQRSTUVWXYZ
QMbABCDEFGHIJKLMNOPQRSTUVWXYZ
QNbABCDEFGHIJKLMNOPQRSTUVWXYZ
QObABCDEFGHIJKLMNOPQRSTUVWXYZ
QPbABCDEFGHIJKLMNOPQRSTUVWXYZ
QQbABCDEFGHIJKLMNOPQRSTUVWXYZ
QRbABCDEFGHIJKLMNOPQRSTUVWXYZ
QSbABCDEFGHIJKLMNOPQRSTUVWXYZ
QTbABCDEFGHIJKLMNOPQRSTUVWXYZ
QUEIAObRBCDFGHJKLMNPQSTUVWXYZ
QVbABCDEFGHIJKLMNOPQRSTUVWXYZ
QWbABCDEFGHIJKLMNOPQRSTUVWXYZ
QXbABCDEFGHIJKLMNOPQRSTUVWXYZ
QYbABCDEFGHIJKLMNOPQRSTUVWXYZ
QZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: R

RbTAOISWCHPMFBEDRLNGUVYJKQZXb
RATLNCIDGbSBPMRVWHYFEUXOZKJAQ
RBIEAObSFUYLRHBCDGJKMNPQTVWXZ
RCEHUIOLYTRAbFKMSBCDGJNPQVWXZ
RDbESILAOYBNRWUCDFGHJKMPQTVXZ
REbSADNCELMFPTGVQIWORBHJUYZKX
RFEOUAILbSBCDFGHJKMNPQRTVWXYZ
RGEAUIOYbHRLSBCDFGJKMNPQTVWXZ
RHAEOYIUbBCDFGHJKLMNPQRSTVWXZ
RINTECSAOGMBVLFPDZKUbJYXHIQRW
RJUEIbABCDFGHJKLMNOPQRSTVWXYZ
RKbESIANLMHPOFTYBCDGJKQRUVWXZ
RLYDIbAESOUBTWCFGHJKLMNPQRVXZ
RMAbSIEYOULFTNCBDGHJKMPQRVWXZ
RNbEAIMSOFLUHDBCPTWYGJKNQRVXZ
ROMUPNDBVLWCSATOGFYRJKbXIHEZQ
RPORbELSHUIATMNBCDFGJKPQVWXYZ
RQUbABCDEFGHIJKLMNOPQRSTVWXYZ
RREIOAYHUbRSNBCDFGJKLMPQTVWXZ
RSbTEOIHAUPYDCKLSBQWFGJMNRVXZ
RTbIAHESYULMORNBFCWJPTDGKQVXZ
RUSCLMETNIGPDBbRZAFOJXYHKQUVW
RVEIAObBCDFGHJKLMNPQRSTUVWXYZ
RWAIEHObBCDFGJKLMNPQRSTUVWXYZ
RXIbESABCDFGHJKLMNOPQRTUVWXYZ
RYbITOSBWMDNLPEACKRUFGHJQVXYZ
RZEIAObBCDFGHJKLMNPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: S

SbATOIWSCBFHNMPDERLGUVYJKQZXb
SAIYMTLRNCGWVFPDBSbKUAHEXJOQZ
SBAEUObYBCDFGHIJKLMNPQRSTVWXZ
SCIORHUAELbYSBCDFGJKMNPQTVWXZ
SDOIAEbUYBCDFGHJKLMNPQRSTVWXZ
SEbNDLSRECATVQPMIFWXGUHYBOZJK
SFUAEOIYRbBCDFGHJKLMNPQSTVWXZ
SGRUOINbABCDEFGHJKLMPQSTVWXYZ
SHOEAbIMUNRSLTCKWYDFHPVBGJQXZ
SIONTSDCBGMVRALZXEFbUPKHQIJWY
SJbABCDEFGHIJKLMNOPQRSTUVWXYZ
SKbEISAYULBCDFGHJKMNOPQRTVWXZ
SLYAIEOUbBCDFGHJKLMNPQRSTVWXZ
SMbAIEOSUYBCDFGHJKLMNPQRTVWXZ
SNEAOTIUCFYbBDGHJKLMNPQRSVWXZ
SObMNCULRPOVFDAIBESTHWJGKQXYZ
SPEOIARLHUbSYBCDFGJKMNPQTVWXZ
SQUbABCDEFGHIJKLMNOPQRSTVWXYZ
SRAEOUSITbBCDFGHJKLMNPQRVWXYZ
SSbIEUAOFMYLNRSHWPBCDGJKQTVXZ
STbIAREOSUMYLHTWNFPCBDGJKQVXZ
SURCBPLMEAGFSNIDObTVKHJQUWXYZ
SVAEbBCDFGHIJKLMNOPQRSTUVWXYZ
SWEIOAUbBCDFGHJKLMNPQRSTVWXYZ
SXbABCDEFGHIJKLMNOPQRSTUVWXYZ
SYSCbMNLRADGOPBEFHIJKQTUVWXYZ
SZTbABCDEFGHIJKLMNOPQRSUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: T

TbTIAOWSHBMCFPDELRNUGYVKJQZXb
TANLTIRKBGCbSFPYMUOXWVEDQZAHJ
TBAOREUbIBCDFGHJKLMNPQSTVWXYZ
TCHYbOAIELRBCDFGJKMNPQSTUVWXZ
TDbOAIBCDEFGHJKLMNPQRSTUVWXYZ
TERDbNSLMEACGXPFVOYWITBUHJZKQ
TFUOIALRYbBCDEFGHJKMNPQSTVWXZ
TGOREUbABCDFGHIJKLMNPQSTVWXYZ
THEAbIORUSYLMWDFNHPCBQGJKTVXZ
TIOCNVMELTSAFGRBZbPDQUIYWHJKX
TJEUbABCDFGHIJKLMNOPQRSTVWXYZ
TKIbABCDEFGHJKLMNOPQRSTUVWXYZ
TLYEAIObLBCDFGHJKMNPQRSTUVWXZ
TMEAOIbBCDFGHJKLMNPQRSTUVWXYZ
TNEAITCbOUNBDFGHJKLMPQRSVWXYZ
TObRONMWLGPTUCSDBXIKEFYAVJHQZ
TPUOALbCIPRBDEFGHJKMNQSTVWXYZ
TQUbABCDEFGHIJKLMNOPQRSTVWXYZ
TRAIUOEYbRVBCDFGHJKLMNPQSTWXZ
TSbEITOMUHYCABKPDFGJLNQRSVWXZ
TTELIAROYbSHMVBNUCDFGJKPQTWXZ
TURDATNSBMEILOPFCGbHQXJKUVWYZ
TVIbABCDEFGHJKLMNOPQRSTUVWXYZ
TWOEIAbBCDFGHJKLMNPQRSTUVWXYZ
TXbABCDEFGHIJKLMNOPQRSTUVWXYZ
TYbPLSMRCAINGBDEFHJKOQTUVWXYZ
TZbEUFHJLABCDGIKMNOPQRSTVWXYZ

DIGRAMS WHERE LEADING LETTER IS: U

UbSAWTHMCDIKBPFGNLOREYUJVXZQb
UALTRGNIDbSBCYMEKVWZAFHJOPQUX
UBLJSTEBVMbIOACDUHRWGNFKPQXYZ
UCHTECIKAOLRUYbBDFGJMNPQSVWXZ
UDEIGYDbAOSLHRUNWBCFJKMPQTVXZ
UEbSNDRLEBTAFMUVPQCGHIJKOWXYZ
UFFAUELIKbBCDGHJMNOPQRSTVWXYZ
UGHGbESUALIMONWBCDFJKPQRTVXYZ
UHbADLTBCEFGHIJKMNOPQRSUVWXYZ
UITRLDSNCEPVbABZFGXHIJKMOQUWY
UJAIUbBCDEFGHJKLMNOPQRSTVWXYZ
UKAEbOILRTUYBCDFGHJKMNPQSVWXZ
ULDATbLESIFOUYMNGPCKBWXHJQRVZ
UMEbBPASMOINCUHFVDGJKLQRTWXYZ
UNDITCGbAESLNRFPKBWJHOUMQVYZX
UOUTDNbRLCIMPSVWABEFGHJKOQXYZ
UPbPOSETRIALHWYBUDGCFJKMNQVXZ
UQUbABCDEFGHIJKLMNOPQRSTVWXYZ
UREbTIASNYROPCGVKBDMFHLUZJQWX
USbTEISLNUCAHPKBYOQDFGJMRVWXZ
UTbIEHTSUOYLRACPBMWGDFNJKQVXZ
UUMNRbABCDEFGHIJKLOPQSTUVWXYZ
UVEAIRObBCDFGHJKLMNPQSTUVWXYZ
UWbABCDEFGHIJKLMNOPQRSTUVWXYZ
UXbUILTEAHBCDFGJKMNOPQRSVWXYZ
UYbIESUABCDFGHJKLMNOPQRTVWXYZ
UZZbOECKMABDFGHIJLNPQRSTUVWXY

DIGRAMS WHERE LEADING LETTER IS: V

VbSCTADEHIWJRBFMPNOYGKLQUVXZb
VALRTNbISGCDBUKPMHJAEFOQVWXYZ
VBbABCDEFGHIJKLMNOPQRSTUVWXYZ
VCObABCDEFGHIJKLMNPQRSTUVWXYZ
VDAbBCDEFGHIJKLMNOPQRSTUVWXYZ
VEbRNSLDMATYIHGEXWFBCVJKOPQUZ
VFbABCDEFGHIJKLMNOPQRSTUVWXYZ
VGbABCDEFGHIJKLMNOPQRSTUVWXYZ
VHbABCDEFGHIJKLMNOPQRSTUVWXYZ
VINDECSOTLRVABGIbUZMKXFHJPQWY
VJbABCDEFGHIJKLMNOPQRSTUVWXYZ
VKbABCDEFGHIJKLMNOPQRSTUVWXYZ
VLAEObBCDFGHIJKLMNPQRSTUVWXYZ
VMbABCDEFGHIJKLMNOPQRSTUVWXYZ
VNbABCDEFGHIJKLMNOPQRSTUVWXYZ
VOLITUCRKYWNGMOSbXABDEFHJPQVZ
VPbABCDEFGHIJKLMNOPQRSTUVWXYZ
VQbABCDEFGHIJKLMNOPQRSTUVWXYZ
VRAOEIbBCDFGHJKLMNPQRSTUVWXYZ
VSbKOABCDEFGHIJLMNPQRSTUVWXYZ
VTbIABCDEFGHJKLMNOPQRSTUVWXYZ
VULEbABCDFGHIJKMNOPQRSTUVWXYZ
VVbABCDEFGHIJKLMNOPQRSTUVWXYZ
VWVbABCDEFGHIJKLMNOPQRSTUWXYZ
VXbABCDEFGHIJKLMNOPQRSTUVWXYZ
VYbAISBCDEFGHJKLMNOPQRTUVWXYZ
VZIbABCDEFGHJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: W

WbTAIOWHSMCPBFDYLERNGUVJKQZXb
WASYRTNLIGKVbBDMXHCWAEFJOPQUZ
WBOAbBCDEFGHIJKLMNPQRSTUVWXYZ
WCAIORbBCDEFGHJKLMNPQSTUVWXYZ
WDbSEORIUABCDFGHJKLMNPQTVWXYZ
WERbLEVDNASIPBYTHCFGJKMOQUWXZ
WFULbABCDEFGHIJKMNOPQRSTVWXYZ
WGbABCDEFGHIJKLMNOPQRSTUVWXYZ
WHIEOAYCbBDFGHJKLMNPQRSTUVWXZ
WITLNSDCFPRGVEMbIKABHJOQUWXYZ
WJbABCDEFGHIJKLMNOPQRSTUVWXYZ
WKIWbESABCDFGHJKLMNOPQRTUVXYZ
WLEbYTASIBCDFGHJKLMNOPQRUVWXZ
WMEASbIPBCDFGHJKLMNOQRTUVWXYZ
WNbESIWFYGHRCNTUABDJKLMOPQVXZ
WORUbNMOLVKBEFWGHACDIJPQSTXYZ
WPOLbABCDEFGHIJKMNPQRSTUVWXYZ
WQbABCDEFGHIJKLMNOPQRSTUVWXYZ
WRIOEAYUbBCDFGHJKLMNPQRSTVWXZ
WSbPOHUIWTYEKMNABCDFGJLQRSVXZ
WTHbABCDEFGIJKLMNOPQRSTUVWXYZ
WUbNRSABCDEFGHIJKLMOPQTUVWXYZ
WVAbBCDEFGHIJKLMNOPQRSTUVWXYZ
WWObABCDEFGHIJKLMNPQRSTUVWXYZ
WXbABCDEFGHIJKLMNOPQRSTUVWXYZ
WYEbABCDFGHIJKLMNOPQRSTUVWXYZ
WZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: X

XbAOTWMISCFPBHRDLYEVUGNKXJQZb
XAMCTLGNSEbABDFHIJKOPQRUVWXYZ
XBbABCDEFGHIJKLMNOPQRSTUVWXYZ
XCELHIRUbABCDFGJKMNOPQSTVWXYZ
XDbABCDEFGHIJKLMNOPQRSTUVWXYZ
XERDCSMbNTQABEFGHIJKLOPUVWXYZ
XFObABCDEFGHIJKLMNPQRSTUVWXYZ
XGbABCDEFGHIJKLMNOPQRSTUVWXYZ
XHIAOUbBCDEFGHJKLMNPQRSTVWXYZ
XISMCNOTBELDAFGRVbHIJKPQUWXYZ
XJbABCDEFGHIJKLMNOPQRSTUVWXYZ
XKbABCDEFGHIJKLMNOPQRSTUVWXYZ
XLEIYbABCDFGHJKLMNOPQRSTUVWXZ
XMbABCDEFGHIJKLMNOPQRSTUVWXYZ
XNEbABCDFGHIJKLMNOPQRSTUVWXYZ
XONRGTCDbABEFHIJKLMOPQSUVWXYZ
XPELROAIbBCDFGHJKMNPQSTUVWXYZ
XQUbABCDEFGHIJKLMNOPQRSTVWXYZ
XRECbABDFGHIJKLMNOPQRSTUVWXYZ
XSbOABCDEFGHIJKLMNPQRSTUVWXYZ
XTEbRHISUYBAOCDFGJKLMNPQTVWXZ
XUARLDOSbBCEFGHIJKMNPQTUVWXYZ
XVbIABCDEFGHJKLMNOPQRSTUVWXYZ
XWEbABCDFGHIJKLMNOPQRSTUVWXYZ
XXVIbABCDEFGHJKLMNOPQRSTUWXYZ
XYGbLPSABCDEFHIJKMNOQRTUVWXYZ
XZbABCDEFGHIJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: Y

YbTAOIWSBCHFDPMRELNGUVYKJQZXb
YALNRbMSKWBGOTEPACDFHIJQUVXYZ
YBOEIRYbABCDFGHJKLMNPQSTUVWXZ
YCHELIAOYbBCDFGJKMNPQRSTUVWXZ
YDREAbINBCDFGHJKLMOPQSTUVWXYZ
YEADSTRbLENBIMOPUWCFGHJKQVXYZ
YFUIEbABCDFGHJKLMNOPQRSTVWXYZ
YGEOIDRbABCFGHJKLMNPQSTUVWXYZ
YHObEABCDFGHIJKLMNPQRSTUVWXYZ
YINESbABCDFGHIJKLMOPQRTUVWXYZ
YJbABCDEFGHIJKLMNOPQRSTUVWXYZ
YKOENbABCDFGHIJKLMPQRSTUVWXYZ
YLELOIVbACUPSYBDFGHJKMNQRTWXZ
YMPEBOAMNIbSCDFGHJKLQRTUVWXYZ
YNTADCOEbGINXUHBFJKLMPQRSVWYZ
YOUNRGbCDTSMEFJKPVWABHILOQXYZ
YPENOIHTRbABCDFGJKLMPQSUVWXYZ
YQbABCDEFGHIJKLMNOPQRSTUVWXYZ
YREIDAOTUbFSBCGHJKLMNPQRVWXYZ
YSbTIEPSMCHOFABDGJKLNQRUVWXYZ
YTHEOIAbBCDFGJKLMNPQRSTUVWXYZ
YUbADKMNSYBCEFGHIJLOPQRTUVWXZ
YVIEWbABCDFGHJKLMNOPQRSTUVXYZ
YWHOARbBCDEFGIJKLMNPQSTUVWXYZ
YXbABCDEFGHIJKLMNOPQRSTUVWXYZ
YYAbBCDEFGHIJKLMNOPQRSTUVWXYZ
YZEAIbBCDFGHJKLMNOPQRSTUVWXYZ

DIGRAMS WHERE LEADING LETTER IS: Z

ZbATWIFHPBESGCDRNOJLMVQKUXYZb
ZATBRbDNIGMSCLYAEFHJKOPQUVWXZ
ZBEbABCDFGHIJKLMNOPQRSTUVWXYZ
ZCAObBCDEFGHIJKLMNPQRSTUVWXYZ
ZDbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZEDbRNSCTAKPLMIUBEFGHJOQVWXYZ
ZFAbBCDEFGHIJKLMNOPQRSTUVWXYZ
ZGbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZHNUEbABCDFGHIJKLMOPQRSTVWXYZ
ZINOEbDMLSABCFGHIJKPQRTUVWXYZ
ZJAbBCDEFGHIJKLMNOPQRSTUVWXYZ
ZKAbBCDEFGHIJKLMNOPQRSTUVWXYZ
ZLEbISABCDFGHJKLMNOPQRTUVWXYZ
ZMAbBCDEFGHIJKLMNOPQRSTUVWXYZ
ZNbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZONRbPOLTEKSABCDFGHIJMQUVWXYZ
ZPbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZQbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZRbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZSbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZTbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZUbREALSBCDFGHIJKMNOPQTUVWXYZ
ZVOEbABCDFGHIJKLMNPQRSTUVWXYZ
ZWEbABCDFGHIJKLMNOPQRSTUVWXYZ
ZXbABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYbMABCDEFGHIJKLNOPQRSTUVWXYZ
ZZLIEAObBCDFGHJKMNPQRSTUVWXYZ

APPENDIX 5

READING LIST

The material within this appendix supplements the reference list as background reading. Sources are listed under the most appropriate chapter in the dissertation, although some sources may have relevance elsewhere.

CHAPTER 2

AYLOR J.H., JOHNSON E.L.
1983
Microcomputing to aid the handicapped part I. IEEE Micro, June, 6-7.

BOWEN C., BOWEN P.
1985
Computing and the disabled. Online Today, April, 14-24.

BRACKENBURY R.
1986
We are physically limited creatures: Disability. Resurgence, 114, 20-21.

DAMPER R.I.
1982
Speech technology Implications for biomedical engineering. Journal of Medical Engineering and Technology, 6:4.

DE GRAAF A.S., RYBNIKAR M.
1986
'Locked-in' but not 'locked-out': A case study. South African Medical Journal, 69, 839-840.

DONNELLY K.
1987
What the "normal" kids learn about disabilities. USA Newspaper.

DUMPER C., NEEN DD.J.
1987
A progression from simple encoding to morse code. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 737,739.

GOODENOUGH-TREPAGNIER C., ROSEN M.J.
1987
Quantification of relative importance of communication needs. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 151-153.

GOODENOUGH-TREPAGNIER C., ROSEN M.J., JANDURA L., GETSCHOW C.O., GENOESE F., FELTS T.
1987
Preliminary validation of prescription guide for selection of communication aids. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 100-102.

HSI S., BARKER M.R., AGOGINO A.M., YAZDANI-KACHOEE B.
1987
Expert systems applied to rehabilitation engineering selection: a new approach to the evaluation of control. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 148-150.

MABRY F., OLSON W.A.
1987
The architecture of an executive workstation for the disabled. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 748-749.

MALONEY S.R.
1983
Impairments, disabilities and handicaps - a comprehensive approach. Rehabilitative Research and Development Centre, Palo Alto, Ca, September.

MULLER A.
1985
Rekenaarbenutting in die onderrig en moontlike beroepsbeoefening van die gehoorgestremde: onderwysbehoeftes. Rehabilitasie in SA, Maart, 19-22.

NEWELL A.F.
1979
Do we know how to design communication aids? Proceedings of the Second International Conference on Rehabilitation Engineering, Ottawa, 345-346.

NEWELL A.F.
1985
Developing appropriate software. Occupational Therapy, 242-243.

NEWELL A.F., ARNOTT J.L.
1985
Introducing microcomputers into occupational therapy - UK experience. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, Te, 200-202.

NEWELL A.F., BROOKS C.P.
1985
The potential of Pitman shorthand transcription as an aid to the deaf. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, Te, 194-196.

NEWELL A.F., DOWNTON A.CC., BROOKS C.P., ARNOTT J.L.
1985
Machine shorthand transcription used as an aid for the hearing impaired and in commercial environments. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 559-560.

ROMSKI M.A., LLOYD L.L., SEVCIK R.A.
1986
Augmentative and alternative communication issues. In Schiefelbusch R.L., Lloyd L.L. (eds.), Language Perspectives: Volume II, 1-45.

ROTHSCHILD N., RYAN S., PARNES P., MILNER M.
1987
Collaboration - an essential issue in the development of augmentative and alternate communication devices. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 91-93.

SCRIMSHAW D.J.

Logo and artistic children. M.I.T. A.I. Laboratory.

SMART S., MACKENSIE L., RICHARDS D.
1985
The design of viable software for use in occupational therapy. Occupational Therapy, 296-298.

THURLOW W.R.
1980
Studies on hand-held visual communication device for the deaf and speech-impaired 2. Keyboard design. Ear and Hearing, 1:3, 141-147.

WEIR S.
1979
The evaluation and cultivation of spatial and linguistic abilities in individuals with cerebral palsy. M.I.T. A.I. Laboratory, Memo: 570.

WINTER J., NEWELL A.F., ARNOTT J.L.
1985
The therapeutic applications of computerised games. International Journal of Bio-Medical Computing, 17, 285-293.

CHAPTER 3

ALM N., NEWELL A.F., ARNOTT J.L.
1987
A communication aid which models conversational patterns. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 127-129.

ARNOTT J.L., PICKERING J.A., SWIFFIN A.L., BATTISON M.
1979
An adaptive and predictive communication aid for the disabled exploits the redundancy in natural language. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 349-350.

BLISS C.K.
1965
Semantography. Sidney, Australia: Semantography Publications.

BRESLER M.I.
1987
Computer aided construction of nonverbal communication boards. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 183-186.

BUHR P.A., HOLTE R.C.
1981
Some considerations in the design of communication aids for the severely physically disabled. Medical and Biological Engineering and Computing, 19, 725-733.

CARLSON G.S., BERNSTEIN J.
1987
Speech recognition of impaired speech. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 103-105.

CARLSON G.S., KLECZEWSKA M., STEELE R., WEINRICH M.
1987
Designing a computerized visual communication system for severe aphasics. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 94-96.

COFFEY J.L.
1961
A comparison of vertical and horizontal arrangements of alpha-numeric material - Experiment 1, Human Factors, 3, 93-98.

COOK A.M., GREY T.L.
1987
Computer aided assessment for the selection of augmentative communication system characteristics. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 142-144.

CRONK S.R., SCHUBERT R.W.
1987
Development of a real-time expert system for automatic adaptation of scanning rates. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 109-111.

DABBAGH H.H., DAMPER R.I.
1985
Average selection length and time as pedictors of communication rate. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, Te., 404-405.

DEMASCO P., HORSTMANN H., ANDRELLOS P., GILBERT M., MINNEMAN S., WOERPEL D.
1987
The implementation of a software methodology for communication aids. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 745-747.

DEMSETZ L.A., ROSEN M.J., GOODENOUGH-TREPAGNIER C.
1983
Communication rate prediction. Proceedings of the Sixth Annual Conference on Rehabilitation Engineering, San Diego, Ca., 156-158.

DOLMAN L., MEEKS S.
1987
Alternative access methods for the IBM family of personal computers and true compatibles. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 678-679.

EDWARDS D.M., EDWARDS C.M.
1987
Effect of assigning standard computer keyboard operating characteristics on quadruplegic input rate. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 702-704.

FRIEDMAN R.B., CHEUNG S., ENTINE S., BARTELL T.
1979
Verbal communication aid for nonvocal patients. Medical and Biological Engineering and Computing, 17, 103-106.

GALYAS K.
1987
Voice output communication aids: a survey and an evaluation. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 97-99.

HECKATHORNE C.W., GORDON R.
1983
The MicroDEC environmental control system: What do users do with it? - A preliminary study. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 603-604.

HECKATHORNE C.W., LEIBOWITZ L.
1985
PACA: Portable anticipatory communication aid. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, Te, 329-330.

HECKATHORNE C.W., LEIBOWITZ L.
1984
The MICRODEC II System - Integrated device control and computer access. Proceedings of the Closing the Gap Conference, 81-87.

HECKATHORNE C.W., LEIBOWITZ L., STRYSIK J.
1983
MICRODEC II - Anticipatory computer input aid. Proceedings of the Sixth Annual Conference on Rehabilitation Engineering, San Diego, Ca., 34-36.

HEHNER B.
1980
Blissymbols for Use. Toronto, Ontario: Blissymmbolics Communication Institute.

HERMAN F.
1985
Music therapy for the young child with cerebral palsy who uses Blissymbols. Music Therapy, 5:1, 28-36.

HUNNICUTT S.
1984
A lexical prediction system. Proceedings of the Third Conference on Augmentative and Alternate Communication, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1-4.

JAKOBSSON M.
1986
Autocompletion in full text transaction entry: A method for humanized input. Proceedings of the CHI Conference, 327-332.

JOHNSON E.L.
1986
Keyboards for the handicapped. Journal of Medical Systems, 10:3, 277-287.

KERR B.
1973
Processing demands during mental operations. Memory and Cognition, 1, 401-412.

KLECZEWSKA M., CARLSON G.S., STEELE R., WEINRICH M.
1987
Patterns of learning in aphasics trained on a computer-based visual communication system. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 157-159.

KORBA L.W., NELSON P.J., PARK G.C.
1987
The MOD keyboard: a progress report. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 699-701.

KRAAT A., DELGADO M., VERA M.
1987
"Yo voy, el va": a case study in integrated Spanish, symbols and a synthetic speech output device. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 168-170.

LOGAN G.D.
1980
Short-term memory demands of reaction-time tasks that differ in complexity. Journal of Experimental Psychology: Human Perception and Performance, 6, 375-389.

MCFARLAND S.R., SCADDEN L.A.
1986
Marketing rehabilitation engineering. SOMA, July, 19-23.

MCLAUGHLIN J., SMITH G.
Blissymbolics: Bliss Skills. Software Description of Bliss Package on Apple.

MILLAR K.
1975
Processing capacity requirements of stimulus coding. Acta Psychologia, 39, 393-410.

NEWELL A.F.
1987
Computer based communication systems - the future. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 171-173.

OGDEN W.C., MARTIN D.W., PAAP K.R.
1980
Processing demands of encoding: What does secondary task performance reflect? Journal of Experimental Psychology: Human Perception and Performance, 6, 355-367.

PAAP K.R., OGDEN W.C.
1981
Letter encoding is an obligatory but capacity-demanding operation. Journal of Experimental Psychology: Human Perception and Performance, 7, 518-527.

ROMICH B., BAKER B.
1987
Application software: a new concept in augmentative and alternative communication. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 154-156.

ROSEN M.J., GOODENOUGH-TREPAGNIER C., FELTS T., GEOESE-ZERBI F.
1987
Quantification of device evaluation. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 180-182.

ROUNDHILL D.N.
A communication device for the physically disabled. Imperial College, London; Tulane University, New Orleans; Children's Hospital, New Orleans.

SCHNEIDER W., SHIFFRIN R.M.
1977
Controlled and automatic human information processing: I. Detection, search and attention. Psychological Review, 84, 1-57.

SCHWARTZ S.P.
1976
Capacity limitations in human information processing. Memory and Cognition, 4, 763-768.

SHAW M.
1978
A capacity allocation model for reaction time. Journal of Experimental Psychology: Human Perception and Performance, 4, 586-598.

SHIFFRIN R.M., SCHNEIDER W.
1977
Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. Psychological Review, 84, 127-187.

SPAETH D.M.
1987
Designing enhancements for and providing technical feedback to manufacturers of augmentative communication equipment. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 136-138.

SWIFFEN A.L., PICKERING J.A., ARNOTT J.L., NEWELL A.F.
1985
PAL: An effort efficient portable communication aid and keyboard emulator. Proceedings of the Eighth Annual Conference on Rehabilitation Engineering, Memphis, Te, 197-199.

VANDERHEIDEN G.C., LEE C.C., SCADDEN L.A.
1987
Features to increase the accessibility of computers by persons with disabilities: report from the industry/government task force. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 750-753.

VODA J., LEIBOWITZ L., WU Y.
1987
Clinical application of an auditory scanning software pprogram with severely disabled, nonspeaking individuals. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 139-141.

ZIPF G.K.
1947
Human Behaviour and the Principle of Least Effort. Cambridge, Mass.: Addison-Wesley Press, Inc.

CHAPTER 4

ALLEN R.B.
1981
Composition and editing of text. Ergonomics, 24, 611-622.

ARNOTT J.L., NEWELL A.F.
1979
Stenotype shorthand and speech synthesis in vocal prosthesis for the dexterous speech impaired. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 621-622.

BAKER B.
1986
Using images to generate speech. Byte, 11:3, 160-168.

BERNSTEIN G., STEVENS M.
1986
Two methods of text entry. SRI International, 3333 Ravenswood Ave., Menlo Park, CA 94025.

BEUKELMAN D., YORKSTON K., POBLETE M., NARANJO C.
1984
Frequency of word occurrence in communication samples produced by adult communication aid users. Journal of Speech and Hearing Disorders, 49:4, 336.

BADDLEY A., CONRAD R., THOMPSON W.E.
1960
Letter structure of the English language. Nature, 186, 414-416.

BRADY M., KELSO D., VANDERHEIDEN G., BUEHMAN D.
1982
A data-based approach to character/syllable/wordsets. Proceedings of the Fifth Annual Conference on Rehabilitation Engineering, Houston, Tx., p. 1.

CALDWELL E.C., PECKHAM P.D., NIX D.H.
1973
Ngram frequency counts. Developmental Psychology, 9, 266-267.

COLBY K.M., CHRISTINAZ D., PARKISON R.C., GRAHAM S., KARPF C.
1981
A word-finding computer program with a dynamic lexical-semantic memory for patients with anomia using an intelligent speech prosthesis. Brain and Language, 14, 272-281.

CROCHETIERE W.J., BALETSA G.S., FOULDS R.A.
1977
An anticipatory display for communication by the severely disabled, non-vocal person. Society for Information Display Digest, 150-151.

DYE R., NEWELL A.F., ARNOTT J.L.
1984
An adaptive editor for shorthand transcription systems. Proceedings of First IFIP Conference on Human-Computer Interaction, London: Sept., vol 2, 92-96.

FOULDS R.
1980
Communication rates for nonspeech expression as a function of manual tasks and linguistic constraints. Proceedings of the International Conference on Rehabilitation Engineering, Toronto: ICRE, 83-87.

FOULDS R., BALETSA G., CROCHETIERE W.
1975
The effectiveness of language redundancy in non-verbal communication. Proceedings of the Conference on Devices and Systems for the Disabled, Philadelphia: Temple University Health Services Centre.

FOULDS R.A., NEWELL A.F., SOEDE M., HUNNICUTT S., ARNOTT J.L., HECKATHORNE C.W., NELSON P.
1987
Research and applications of lexical prediction in augmentative communication - A morning seminar. Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca.

GIBLER C.D., CHILDRESS D.S.
1983
Adaptive dictionary for computer-based communication aids. Proceedings of the Sixth Annual Conference on Rehabilitation Engineering, San Diego, Ca., 165-167.

GOODENOUGH-TREPAGNIER C.
1980
The development of a non-vocal communication system for pre-reading children. Tufts - New England Medical Center, Rehabilitation Medicine, 3-42.

GOODENOUGH-TREPAGNIER C.
1982
Development of a technique to produce efficient, non-limiting language representation systems for non-vocal communication. International Journal of Rehabilitation Research, 5:3, 391-392.

GOODENOUGH-TREPAGNIER C.
1982
Instructions for making a WRITE board. Tufts - New England Medical Center.

GOODENOUGH-TREPAGNIER C.
1985
Description of lists and layouts. Tufts - New England Medical Center.

GOODENOUGH-TREPAGNIER C., FRIED-OKEN M.
1983
Prespeec: Use of a sound-combination system by non-vocal children, Proceedings of the Sixth Annual Conference on Rehabilitation Engineering, San Diego, Ca., 173-175.

GOODENOUGH-TREPAGNIER C., GOLDENBERG E.P., FRIED-OKEN M.
1981
Nonvocal communication system with unlimited vocabulary using apple and speech-syllables. Fourth Annual Conference on Rehabilitation Engineering, Washington D.C., 173-175.

GOODENOUGH-TREPAGNIER C., PRATHER P.
1981
Communication systems for the nonvocal based on frequent phoneme sequences. Journal of Speech and Hearing Research, 24, 322-329.

GOODENOUGH-TREPAGNIER C., ROSEN M.
1981
Model for a computer-based procedure to prescribe an optimal "keyboard". Proceedings of the Fourth Annual Conference on Rehabilitation Engineering, Washington D.C., 197-199.

GOODENOUGH-TREPAGNIER C., ROSEN M.
1982
Optimal language menu for a one-switch nonvocal communication device. Proceedings of the Fifth Annual Conference on Rehabilitation Engineering, Houston, Tx., p. 2.

GOODENOUGH-TREPAGNIER C., ROSEN M.J., GALDIERI B.

Word menu reduces communication rate.

GOODENOUGH-TREPAGNIER C., TARRY E., PRATHER P.
1982
Derivation of an efficient non-vocal communication system. Human Factors, 24:2, 163-172.

GOODENOUGH-TREPAGNIER C., TARRY E., PRATHER P.
1982a
Derivation of an efficient non-vocal communication system. Human Factors Journal, 1-23a.

HERDEN R.E.
1950
The numerical expression of selective variation in the vowel-consonant sequence in English and Russian. In E. Pulgram (ed.), Studies Presented to Joshua Whatmough, Gravenhage: Mouton & Co., 91-104.

HIGGINS J.M.

Systematic approaches to vocabulary selection for communication aid users. Prentke Romich Company, Wooster, Ohio, pp. 1-13.

LEEDHAM C.G., DOWNTON A.C., BROOKS C.P., NEWELL A.F.
1987
On-line aquisition of Pitman's handwritten shorthand as a means of rapid data entry. 86-91.

LEVINE S.H. GOODENOUGH-TREPAGNIER C., GETSCHOW C.O., MINNEMAN S.L.
1987
Multi-character key text entry using computer disambiguity. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 177-179.

LEVINE S.H., GOODENOUGH-TREPAGNIER C., ROSEN M.J., GETSCHOW C.O.

Adaptive technique for customised interface design with application to non-vocal communication.

MILLIKEN D.D.
1943
Elementary Cryptography and Cryptanalysis, New York: New York University Press.

MINNEMAN S.L.
1986
Keyboard optimization technique to improve output rate of disabled individuals. Tufts - New England Medical Center.

NEWELL A.F., SOEDE M.
1986
Efficient input systems for augmentative communication: Annotated report of a workshop, Institute for Rehabilitation Research, Hoensbroek, The Netherlands.

OLSON D.W., JASINSKI L.E.
1986
Keyboard efficiency. Byte, February, 241-244.

PRATT F.
1939
Secret and Urgent: The Story of Codes and Ciphers, Indianapolis: Bobbs-Merrill, 252-278.

ROBERTS A. HOOD
1965
A Statistical Linguistic Analysis of American English, The Hague: Mouton & Co.

ROSEN M.J., GOODENOUGH-TREPAGNIER C.
1981
Factors affecting communication rate in nonvocal systems. Proceedings of the Fourth Annual Conference on Rehabilitation Engineering, Washington D.C., 194-196.

ROSEN M.J., GOODENOUGH-TREPAGNIER C.
1982
The influence of scan dimensionality on non-vocal communication rate. Proceedings of the Fifth Annual Conference on Rehabilitation Engineering, Houston, Tx, 4.

ROWLEY B.A.
1987
RPM for accessing large vocabulary files. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 165-167.

SHANNON C.E.
1948
A mathematical theory of communication. Bell System Technical Journal, 27, 379-413.

SOEDE M.
1986
Invoermethoden, ofwel: Hoe kan een communicatie apparaaat optimaal bediend worden. Workshop: "Handicap & Communicatie, Delft, 2-7.

SOEDE M., CREMERS G., VAN KNIPPENBERG F., OOSTINJEN E.
1984
Interimrapport: Toepassing Spraaktechnologie de Spraak-gestuurde rolstoel. Institute for Rehabilitation Research, Hoensbroek, The Netherlands.

SOLSO R.L., JUEL C.L.
1980
Positional frequency and versatility of bigrams for two- through nine-letter English words. Behaviour Research Methods and Instrumentation, 12, 297-343.

SOLSO R.L., KING J.F.
1976
Frequency and versatility of letters in the English language. Behaviour Research Methods and Instrumentation, 8, 283-286.

SPRAGUE C.R., SPRAGUE L.G.
1983
Equipment-free communication for the severely disabled. Proceedings of the Six Annual Conference on Rehabilitation Engineering, San Diego, Ca., 174-176.

TOPPER G., MACEY W., SOLSO R.L.
1973
Bigram versatility and bigram frequency. Behaviour Research Methods and Instrumentation, 5, 51-52.

TSU YIH HUANG J.
1987
An interactive Cobol editing system: keystroke reduction application for disabled students. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 708-710.

VAN BALKOM H.L.M., SOEDE M.
1987
Linguistic efficient communication systems; consideration of feedback principles. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 133-135.

VANDERHEIDEN G.C.
1979
A high-efficiency flexible keyboard input acceleration technique: Speedkey. Proceedings of the Second Annual Conference on Rehabilitation Engineering, Ottawa, 353-354.

WEINRICH M., STEELE R.D., KLECZEWSKA M., CARLSON G.S., BAKER E.
1987
Representations of "verbs" in a computerized visual communication system. Proceedings of the Tenth Annual Conference on Rehabilitation Engineering, San Jose, Ca., 162-164.

WHITEFIELD A.
1986
Human factors aspects of pointing as an input technique in interactive computer systems. Applied Ergonomics, 17, 2, pp. 97-104.

WHORF B.L.
1940
Linguistics as an exact science. Technology Review, 43:2, 3-8.

CHAPTER 5

ABERCROMBIE D.
1973
Parameters and phonemes. In: Phonetics in Linguistics: A book of readings. Eds. Jones W.E., Laver J., Longman: London, 14-18.

ARNOLD G.F.
1973
Concerning the theory of plosives. In: Phonetics in Linguistics: A book of readings. Eds. Jones W.E., Laver J., Longman: London, 29-33.

ATKINSON, KILBY, ROCA
1982
Syntax. In: Foundations of General Linguistics. 145-174.

BUCKINGHAM H.W.
Neuropsychological models of language. In: Normal Processes, 323-347.

CARROLL J.B.
1938
Diversity of vocabulary and the harmonic series law of word-frequency distribution. Psychological Record, 2, 377-386.

CARROLL J.B.
1966
Word-frequency studies and the logonormal distribution. In E. Zale (ed.), Proceedings of the Conference on Language and Language Behavior, N.Y.: Appleton-Century-Crofts, 213-235.

DRIEMAN G.
1962
Differences between written and spoken language: An exploratory study I & II. Acta Psychologia Amsterdam, 20, 36-57, 78-100.

FINKBEINER A.
1984
How did language begin. Science, 24-25.

FRIES C.P.
1952
Function words. In The Structure of English. London: Longman, 87-109.

FRY D.B.
1960
Linguistic theory and experimental research. Transactions of the Philological Society, 13-39.

GIBSON J., GRUNER C., KIBLER R., KELLY F.
1966
A quantitative examination of differences and similarities in written and spoken messages. Speech Monographs, 33, 444-451.

GOFFMAN E.
1981
Forms of Talk. Oxford: Blackwell.

HALLE M.
1954
The strategy of phonemics. Word, 10, 197-209.

HARRIS Z.S.
1951
Methods in structural linguistics, Chicago.

HARRIS Z.S.
1954
Distributional structure. Word, 10, 197-209.

HORROWITZ M., NEWMAN J.
1964
Spoken and written expression: An experimental analysis. Journal of Abnormal and Social Psychology, 68, 640-647.

JELINEK F.
Markov source modeling of text generation. Continuous Speech Recognition Group, IBM T.J. Watson Research Center, 2-30.

LYONS J.
1968
The structure of language. In Introduction to Theoretical Linguistics. Cambridge, Mass.: Cambridge University Press, 70-98.

MILLER G.A., NEWMAN E.B.
1958
Tests of a statistical explanation of the rank-frequency relation for words in written English. American Journal of Psychology, 27, 209-258.

MILLER G.A., NEWMAN E.B., FRIEDMAN E.A.
1958
Length-frequency statistics for written English. Information and Control, 1, 370-389.

PIKE K.L.
1947
Phonemics: A Technique for Reducing Languages to Writing. Ann Arbor: University of Michigan Press.

VENEZKY R.L.
1970
The Structure of English Orthography. The Hague: Mouton.

ZIPF G.K.
1935
The Psycho-Biology of Language, Boston: Houghton Mifflin Co.

ZIPF G.K.
1942
Children's speech. Science, 96, 344-345.

CHAPTER 6

GEORGE A.
1983
Philosophy and the birth of Computer Science. Robotics Age, Jan/Feb, 26-31.

GIMPEL J.F.
1986
Processing strings in SNOBOL4. Byte, February, 175-186.

KIMBRELL R.E.
1985
English Recognition. Byte, December, 125-140.

LEIGH W., EVANS J.
1986
Interpretation of natural language database queries using optimization methods. IEEE Trans-actions on Systems, Man and Cybernetics, 16:1, 40-52.

MACKAY P.A.
1986
Typesetting problem scripts. Byte, February, 201-217.

MCKEAN K., DWORETZKY T.
1985
Fuzzy means to logical ends. Discover, February, 70-73.

NEWMAN M.
1986
Poetry processing. Byte, February, 221-228.

POLLACK J., WALTZ D.L.
1986
Interpretation of natural language. Byte, February, 189-198.

TANKARD J.
1986
The literary detective. Byte, February, 231-238.

VOSE G.M., WILLIAMS G.
1986
Computer science considerations. Byte, February, 169-172.

APPENDIX 6

PROGRAM LISTINGS

Appendix 6a - Program Listings

```
;;=================================================================
;; FILE:  LOADUP
;;
;; Bootstraping procedures.
;;=================================================================


(SEND *TERMINAL-IO* :CLEAR-SCREEN)

(PRINC "Please be patient while APRESTO is loaded.")

(DEFUN LOAD-FILE (FILE-NAME)
   (SETF *LOAD-VERBOSE* NIL)
   (LOAD FILE-NAME)
   (PRINC "."))

(MAPCAR 'LOAD-FILE '(EDIT.EVL
                     INIT.EVL
                     MAIN.EVL
                     READ.EVL
                     UTIL.EVL
                     FILEIO.EVL
                     ACCESS.EVL
                     CURSOR.EVL
                     RUNOUT.EVL
                     SCREEN.EVL
                     RULEBAS.EVL
                     SCANNER.EVL
                     RULEMANI.EVL))

(AUTOLOAD REPORT-STATISTICS   "STATS.EVL"     "\\LISP\\APRESTO\\")
(AUTOLOAD READ-IN-DIGRAM-NAME "READDIG.EVL"   "\\LISP\\APRESTO\\")
(AUTOLOAD INITIATE-SIMULATION "SIMULAT.EVL"   "\\LISP\\APRESTO\\")
(AUTOLOAD CHECK-DIGRAM-FILE   "CHECKDIG.EVL" "\\LISP\\APRESTO\\")
(AUTOLOAD DEVALUE-GLOBAL      "DEVALUE.EVL"   "\\LISP\\APRESTO\\")

  (SCANNER)
```

Appendix 6b - Program Listings

```
;;=============================================================================
;;FILE NAME:  MAIN
;;  This file contains the functions which constitute the main procedures
;;  of the evaluator.
;;=============================================================================

(DEFUN SIMULATION-MODE-P NIL
  (EQUAL *PROCESSING-MODE* 'SIMULATION))              ; Return TRUE/NIL

(DEFUN DETERMINE-MODE NIL
  (SCREEN-BOTTOM *MESS-QUERY-3*)
  (COND ((YES-NO-P) (SETQ *PROCESSING-MODE* 'FREE)); If Y, scan as normal
        (T (SETQ *PROCESSING-MODE* 'SIMULATION)))  ; Else simulation
  (REMOVE-BOTTOM))                                  ; Clear up

(DEFUN DETERMINE-MODIFY NIL
  (SCREEN-BOTTOM *MESS-QUERY-11*)
  (COND ((YES-NO-P) (SETQ *RULE-MODIFY* T))         ; Set flag if Yes
        (T (SETQ *RULE-MODIFY* NIL)))               ; Else no rule modification
  (REMOVE-BOTTOM))                                  ; Clear up

(DEFUN DETERMINE-RULE-BASE NIL
  (COND ((KEEP-CURRENT-RULE-BASE) T)                ; If current base okay =>
        (T (CREATE-NEW-BASE))))                     ; Else new base

(DEFUN ASK-RESTART NIL
  (CLEAR-SCREEN *TERMINAL-IO*)                      ; Clear the screen and
  (SCREEN-BOTTOM *MESS-QUERY-4*)                    ; Ask to restart SCANNER
  (DEVALUE-GLOBAL)                                  ; Global variables devalued
  (COND ((YES-NO-P) (INITIALISE-GLOBAL)             ; Initialise
                    (INITIALISE-GROUPS)             ; Initialise groups
                    (EXECUTE-SCANNER))              ; Accept -> rerun SCANNER
        (T (CLEAR-SCREEN *TERMINAL-IO*))))          ; Else clear the screen

(DEFUN EXECUTE-SCANNER NIL
  (OPEN-RUN-FILES)                                  ; Open files for run
  (COND ((DETERMINE-RULE-BASE)                      ; Rule base read in? Go on
         (COND ((NULL *TECHNIQUE*)                  ; Check technique, if NIL
                (REPORT-NO-TECHNIQUE))              ;   Report error =>
               (T (INITIATE-RULES)                  ; Else, Initiate rules
                  (DETERMINE-MODE)                  ; Determine processing mode
                  (DETERMINE-MODIFY)                ; Determine rule-modify
                  (COND ((SIMULATION-MODE-P)
                         (INITIATE-SIMULATION)) ; Start simulation
                        (T (WRITE))))))))           ; Free text -> Write text
        (T ))                                       ; No rule base => restart
  (CLOSE-RUN-FILES)                                 ; Close files used for run
  (ASK-RESTART))                                    ; Ask to restart

(DEFUN SCANNER NIL
  (CLOSE-ALL-FILES)
  (INITIALISE-GLOBAL)                               ; Initialise
  (INITIALISE-GROUPS)                               ; Initialise groups
  (DISPLAY-TITLE)                                   ; Clear screen and display
  (EXECUTE-SCANNER)                                 ; Run scanner
  (DISPLAY-END)                                     ; Clear screen and display
  (DEVALUE-FUNCTIONS))                              ; Devalue all functions
```

Appendix 6c - Program Listings

```
;;=============================================================================
;;FILE NAME:  EDIT
;;
;;  This group of functions control the text processing and the
;;  "intelligence" of the processor.
;;=============================================================================

;;=============================================================================
;;  This group of functions control the actual screen printing.
;;=============================================================================

(DEFUN PRINT-CHOICE (CHOICE)
  (COND ((NULL CHOICE) NIL)                              ; No choice -> return NIL
        (T                                               ; Else ->
     (DELETE-LAST-CHAR *EDIT-WINDOW* 0)                  ; Remove cursor
     (DISPLAY-XY CHOICE (CURRENT-X) (CURRENT-Y)          ; Print choice
                        *EDIT-WINDOW*)                   ;    on edit screen
     (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)             ; Note element is printed
     (FORMAT  *NEW-TEXT-FILE* CHOICE)                    ; Update file
     (INCF *TOTAL-ELEMENTS*)                             ; Increment element counter
     (MOVE-ON (LENGTH CHOICE))                           ; Update screen coordinates
     (COND ((EQUAL CHOICE *BLANK*)                       ; If blank
               (SETQ *LASTWORD* *NEWWORD*)               ;        Save word
               (SETQ *NEWWORD*))                         ;        Reset
           (T (STRING-APPEND *NEWWORD* *BLANK*)))
     (COND ((SIMULATION-MODE-P)                          ; If simulation
               (UPDATE-SIMULATION-FILE CHOICE))          ; Then update that file
           (T )))))                                      ; Else nothing

(DEFUN EXAMINE-STATUS NIL
  (COND ((EQUAL (GET-ENVIRONMENT :STATUS)                ; Element not printed
                'NOT-PROCESSED)
                (PRINT-CHOICE                            ;     => print element
                    (GET-ENVIRONMENT :ELEMENT)))
        (T NIL)))

;;=============================================================================
;;  This group of functions control special text processing.
;;=============================================================================

(DEFUN NEW-LINE NIL
  (REMOVE-CURSOR (CURRENT-X) (CURRENT-Y))                ; Remove cursor
  (TERPRI *EDIT-WINDOW*)                                 ; Advance cursor
  (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)                ; Note element is printed
  (INCF *TOTAL-ELEMENTS*)                                ; Increment element counter
  (COND ((SIMULATION-MODE-P)                             ; If in simulation mode
            (SEND *SIMULATION-FILE*                      ;    then
                  :READ-CHAR-NO-HANG))                   ;       retrieve next char
        (T ))                                            ;    else nothing
  (ADVANCE-LINE)                                         ; Update screen coordinates
  (TERPRI *NEW-TEXT-FILE*)                               ; Send end-of-line to file
  (POSITION-CURSOR (CURRENT-X) (CURRENT-Y)               ; Show cursor
                   *EDIT-WINDOW*))
```

```
(DEFUN DELETE-CHARACTER NIL
  (INCF *KEYSTROKES-DEL*)                          ; Increment counter
  (REINSTATE-ENVIRONMENT *ENV-STORE*)              ; Backtrack environment
  (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)          ; Note status
  (COND ((EQUAL (LENGTH *XY-COORDS*) 1))           ; If no text -> do nothing
    (T (REMOVE-CURSOR (CURRENT-X) (CURRENT-Y))     ; Remove cursor
       (DELETE-LAST-CHAR *EDIT-WINDOW*             ; Else erase character
             (LENGTH (GET-ENVIRONMENT :ELEMENT)))  ;  by retracking length
       (UPDATE-ENVIRONMENT :ELEMENT NIL))))        ;  Update

;;=============================================================================
;;  This group of functions controls the special editing functions of
;;  the editor.
;;=============================================================================

(DEFUN PRINT-FN NIL
   (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)          ; Note element is printed
   (SCREEN-BOTTOM *MESS-PRINT-1*)
   (COND ((EQUAL (SCAN 'YES-NO-GROUP) *NO*)         ; Check for escape
          (SCREEN-BOTTOM *MESS-PRINT-2*))           ; Message
         (T (COND
              ((NULL (OPEN-PRINTER *TEXT-FILE*) ))  ; If printer fails =>
              (T (SCREEN-BOTTOM *MESS-PRINT-3*)     ; Else print
                 (PRINT-TEXT *NEWTEXT* *TEXT-FILE*) ; Print text
                 (TERPRI *TEXT-FILE*)               ; Flush printer
                 (CLOSE *TEXT-FILE*)                ; Disconnect printer
                 (SCREEN-BOTTOM *MESS-PRINT-4*)))))) ; Message
   (REMOVE-BOTTOM))                                 ; Remove message line

(DEFUN SAVE-FN NIL
 )

;;=============================================================================
;;  This group of functions controls the "intelligence" of the editor.
;;=============================================================================

(DEFUN CHECK-CONTINUATION (REST-OF-RULES)
  (COND ((GET-ENVIRONMENT :CONTINUE)                ; If to continue ->
         (COND ((NULL REST-OF-RULES)                ;   If no more rules ->
                (CASCADE-THRU-RULES *RULE-BASE*))   ;     start again
               (T                                   ;   Else
            (CASCADE-THRU-RULES REST-OF-RULES))))   ; rest
        (T ))                                       ; Else do nothing
```

Appendix 6c - Program Listings

```
(DEFUN CHECK-MODIFICATION NIL
   (DISPLAY-XY *MESS-PROMPT-12* 0 *MAXMESSAGEROW*
                                 *MESSAGE-WINDOW*) ; Prompt for interruption
   (COND ((DO ((COUNT 1 (INCF COUNT))
               (KEY (CHECK-KEYPRESS)
                            (CHECK-KEYPRESS)))     ; check for key
              ((OR KEY                             ; Stop if key pressed or
                   (NOT (SIMULATION-MODE-P))       ;     if in free mode or
                   (EQUAL COUNT *RULE-DELAY*))     ;     if end of wait period
                KEY))                              ; return result
       (CLEAR-SCREEN *RULE-WINDOW*)                ; Clear message
       (COND ((MODIFY-RULE (COMPUTE-LEVEL RULE     ; If rules changed =>
                            *RULE-BASE*) NIL)
              (CLEAR-SCREEN *RULE-WINDOW*)         ; Clear message
              (REINSTATE-ENVIRONMENT ENV)          ; reinstate
              (RESET-CONTINUE)
              (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)
              NIL)                                 ; flag change
             (T (CLEAR-SCREEN *RULE-WINDOW*)       ; Clear message
                'T)))                              ; Else continue
         (T  (CLEAR-SCREEN *RULE-WINDOW*) 'T)))    ; Clear message

(DEFUN PROCESS-RULE (RULE)
  (LET ((ENV (NOTE-ENVIRONMENT))                   ; Store current environment
        (RESULT T))                                ; Result storage
    (UPDATE-ENVIRONMENT :RULE RULE)                ; Update environment
    (MAPCAR 'EVAL (CDR RULE))                      ; Evaluate consequence
    (COND (*RULE-MODIFY* (CHECK-MODIFICATION))     ; Modify ?
          (T RESULT))))                            ; Return result

(DEFUN TESTRULE (RULE)
  (EVAL RULE))                                     ; Evaluate rule

(DEFUN CASCADE-THRU-RULES (ASSERTIONS)             ; Find rule in assertions
  (COND ((NULL ASSERTIONS) NIL)                    ; No fit -> return NIL
    (T (COND                                       ; Else
         ((MEMBER (CAR ASSERTIONS)                 ; If rule already tried ->
                  (GET-ENVIRONMENT :RULE) :TEST 'EQUAL)
            (CASCADE-THRU-RULES (CDR ASSERTIONS))) ;  try rest of rules
         ((TESTRULE (CAAR ASSERTIONS))             ; Else evaluate assertion
              (PROCESS-RULE (CAR ASSERTIONS))      ; and if it fits -> process
              (CHECK-CONTINUATION (CDR ASSERTIONS))
              (EXAMINE-STATUS))                    ; Check need for printing
         (T (CASCADE-THRU-RULES (CDR ASSERTIONS))  ;   Else try rest of rules
            (EXAMINE-STATUS))))))                  ;    and check for printing

(DEFUN SIFT-RULES NIL
  (UPDATE-ENVIRONMENT :RULE NIL)
  (CASCADE-THRU-RULES *RULE-BASE*))
```

```
;;=============================================================================
;;  This group of functions control the bottom-level text processing.
;;=============================================================================

(DEFUN REPEAT-SYMBOL NIL
  (COND ((KEYPRESSED (GET-ENVIRONMENT :GROUP)) NIL); Stop when key pressed
        (T
    (COND ((PROCESS-RULE
                  (CAR (GET-ENVIRONMENT :RULE)))) ; Reprocess last rule
          (T (PRINT-CHOICE                        ; Else, if no rule ->
                  (GET-ENVIRONMENT :ELEMENT))))   ;   reprint element
    (REPEAT-SYMBOL))))                            ; Reevaluate

(DEFUN UPDATE-ALL (ELEMENT)
  (FORMAT *MIRRORED-TEXT-FILE*                    ; Update file
      (COND ((STRINGP ELEMENT) ELEMENT)          ; But check that element is
            (T ""))))                             ;   a string and not nil
  (COND ((NULL ELEMENT) (REMEMBER-GROUP)          ; No element -> note group
         (UPDATE-ENVIRONMENT :ELEMENT ELEMENT)) ; Update environment
        (T                                        ; Else
    (COND ((EQUAL ELEMENT *REPEATSYM*) (REPEAT-SYMBOL))
          (T (CLEAR-LIST *GROUPS-NOT-WANTED*)     ; Reset unused groups
             (UPDATE-ENVIRONMENT :RULE NIL)       ; Reset rule to nil
             (UPDATE-ENVIRONMENT :ELEMENT ELEMENT) ; Update environment
           )))))

(DEFUN PROCESS-ELEMENT (ELEMENT)
  (SETQ *CURRENT-YES* 0)                          ; Initialise counters
  (SETQ *CURRENT-NO*  0)
  (UPDATE-ALL ELEMENT)                            ; Process updates
  (SIFT-RULES))                                   ; Sift through rules

(DEFUN WRITE-ONE-UNIT NIL
   (COND ((OR (GROUP-ATTEMPTED-P (GET-ENVIRONMENT :NAME))
              (EQUAL (GET-ENVIRONMENT :NAME) NIL))
                  (SIFT-RULES))
         (T (PROCESS-ELEMENT (SCAN                ;       Else Process chosen
                  (GET-ENVIRONMENT :GROUP)))))))  ;        from current group

;;=============================================================================
;;  Implement writing as a three-tier function to simulate recursion without
;;  possibility of a stack reset.
;;=============================================================================

(DEFUN WRITE-ONE-LINE NIL
  (DO ((LOOP 0 (INCF LOOP)))                      ; Loop until
      ((OR (EQUAL LOOP *MAXEDITCOL*)              ;   loop = line
           *QUIT*  ))                             ;   or end -> return
    (COND ((SIMULATION-MODE-P) (SIMULATE-ONE-UNIT)); Use simulation
          (T (WRITE-ONE-UNIT)))) )                ; or free mode

(DEFUN WRITE-ONE-PARA NIL
  (DO ((LOOP 0 (INCF LOOP)))                      ; Loop until
      ((OR (EQUAL LOOP *MAXEDITPARA*)             ;   loop = para
           *QUIT* ) )                             ;   or end -> return
    (WRITE-ONE-LINE)))                            ;  write a "line" at a time
```

Appendix 6c - Program Listings

```
(DEFUN WRITE-ONE-PAGE NIL
  (DO ((LOOP 0 (INCF LOOP)))                    ; Loop until
      ((OR (EQUAL LOOP *MAXEDITROW*)            ;   loop = page
           *QUIT* ) )                           ;   or end -> return
    (WRITE-ONE-PARA)))                          ;  write a "para" at a time

(DEFUN WRITE NIL
  (SETQ *TIME* (GET-TIME))                      ; Get time
  (DO ((LOOP 0 (INCF LOOP)))                    ; Loop until
      ((OR (> LOOP *MAXEDITPAGE*)               ;   loop > max allowed
           *QUIT*) (REPORT-STATISTICS)  )       ;   or end -> statistics
    (WRITE-ONE-PAGE)))                          ;  write a "page" at a time
```

```
;;=============================================================================
;;FILE NAME:  INIT
;;
;;  This group of functions define the initialising of global constants,
;;  parameters and variables.  A function to devalue these global constructs
;;  is also defined.  The basic rule-base is initialised to include basic
;;  time-saving rules.
;;=============================================================================

(DEFUN INITIALISE-GLOBAL NIL

  (DEFINE-CONSTANTS)
  (DEFINE-MESSAGES)
  (DEFINE-PARAMETERS)
  (DEFINE-VARIABLES)
  (DEFINE-ENVIRONMENT)
  (DEFINE-BASIC-RULES))

(DEFUN DEFINE-CONSTANTS NIL

  ;; Define the following constants to be used in the environment:
  ;;          (may not be changed during program)

  ;; Define specific constants:

  (DEFCONSTANT *BASE-DELAY*          2)          ; Fastest delay factor
  (DEFCONSTANT *WAIT-DELAY*       4000)          ; Wait period for delay
  (DEFCONSTANT *RULE-DELAY*        150)          ; Wait period for rule edit
  (DEFCONSTANT *BASEX*               0)         ; Value of base coord. X
  (DEFCONSTANT *BASEY*              23)          ; Value of base coord. Y
  (DEFCONSTANT *MAXEDITPAGE*        20)          ; Maximum pages of edit
  (DEFCONSTANT *MAXEDITPARA*        20)          ; Maximum paragraphs
  (DEFCONSTANT *MAXRULEROW*          4)          ; Maximum rows rule window
  (DEFCONSTANT *MAXRULECOL*         79)          ; Maximum cols rule window
  (DEFCONSTANT *MAXMESSAGEROW*       3)          ; Maximum rows for messages
  (DEFCONSTANT *MAXMESSAGECOL*      79)          ; Maximum cols for messages
  (DEFCONSTANT *MAXSCRNCOL*         79)          ; Maximum screen columns
  (DEFCONSTANT *SCREENLENGTH*       24)          ; Length of screen
  (DEFCONSTANT *WORD-LENGTH*         6)          ; Length of a word
  (DEFCONSTANT *PERIOD*     (STRING #\.))        ; Full stop character
  (DEFCONSTANT *BLANK*      (STRING #\ ))        ; Blank character
  (DEFCONSTANT *NULSYM*     (STRING #\#))        ; Null character
  (DEFCONSTANT *CURSOR*     (STRING 219))        ; Cursor character
  (DEFCONSTANT *QUITSYM*    (STRING 234))        ; Quit symbol
  (DEFCONSTANT *INSERTSYM*  (STRING  16))        ; > symbol
  (DEFCONSTANT *DELETESYM*  (STRING  17))        ; < symbol
  (DEFCONSTANT *CARET*      (STRING #\^))        ; ^ symbol
  (DEFCONSTANT *ASTERISK*   (STRING  42))        ; * symbol
  (DEFCONSTANT *PUNCTSYM*   (STRING 168))        ; Punctuation symbol
  (DEFCONSTANT *REPEATSYM*  (STRING 236))        ; Repeat symbol
  (DEFCONSTANT *PRINTSYM*   (STRING 158))        ; Print symbol
  (DEFCONSTANT *QUESMARK*            '?)         ; Question mark
  (DEFCONSTANT *YES*              "YES")         ; Yes word
  (DEFCONSTANT *NO*                "NO")         ; No word
  (DEFCONSTANT *ESC*              "ESC")         ; Esc word
  (DEFCONSTANT *ESCSYM*             27)          ; Esc symbol
```

```
(DEFCONSTANT *SPEEDUP*     (STRING-APPEND
                                    '"$" '"+"))     ; $+ symbol
  (DEFCONSTANT *SPEEDDOWN*  (STRING-APPEND
                                    '"$" '"-"))     ; $- symbol
  (DEFCONSTANT *RETURNSYM*  (STRING-APPEND
                                  17 205 190))      ; Carriage return
  (DEFCONSTANT *EOL*                  10)           ; End of line
  (DEFCONSTANT *EOF*                 NIL)           ; End of file
  (DEFCONSTANT *ENDDIG*     (STRING   42))          ; End of digram ("*")
  (DEFCONSTANT *BEGDIG*     (STRING   93))          ; Begining of digram ("]")
  (DEFCONSTANT *LOWU*       (STRING #\u))           ; Lower case "u"
  (DEFCONSTANT *UPPU*       (STRING #\U))           ; Upper case "u"
  (DEFCONSTANT *QSET* (LIST (STRING #\Q)            ; Set of q's
                            (STRING #\q)))
  (DEFCONSTANT *PUNCTUATION* (STRING #\,))          ; Comma character
  (DEFCONSTANT *SENTENCE-END* (LIST                 ; Set of punctuation marks
        (STRING #\.) (STRING #\?) (STRING #\!)))
  )

(DEFUN DEFINE-MESSAGES NIL

  ;; Define the following screen messages to be used in the environment:
  ;;          (may not be changed during program)

                    ; Printing messages
                    ; ==================

  (DEFCONSTANT *MESS-PRINT-1*
     "Are you sure that the printer is on.")
  (DEFCONSTANT *MESS-PRINT-2*
     "Printing has been stopped - press switch to continue.")
  (DEFCONSTANT *MESS-PRINT-3*
     "Printing in progress.....")
  (DEFCONSTANT *MESS-PRINT-4*
     "Printing completed - press switch to continue.")

                     ; Query messages
                     ; ==============

  (DEFCONSTANT *MESS-QUERY-1*
     "Is the above information correct?  (Y/N)")
  (DEFCONSTANT *MESS-QUERY-2*
     "Would you like to halt this procedure?  (Y/N)")
  (DEFCONSTANT *MESS-QUERY-3*
     "Do you want to scan in free mode?  (Y/N)")
  (DEFCONSTANT *MESS-QUERY-4*
     "Would you like to return to APRESTO? (Y/N) ")
  (DEFCONSTANT *MESS-QUERY-5*
     "Would you like to save this run? (Y/N) ")
  (DEFCONSTANT *MESS-QUERY-6*
     "Would you like to print this run? (Y/N) ")
  (DEFCONSTANT *MESS-QUERY-7*
     "Check that printer is on and then press any key.")
  (DEFCONSTANT *MESS-QUERY-8*
     "Would you like to see the history? (Y/N) ")
  (DEFCONSTANT *MESS-QUERY-9*
     " technique has been initialised.~%    Do you want to continue? (Y/N) ")
```

```
  (DEFCONSTANT *MESS-QUERY-10*
     " already exists.~%   Do you want to erase the existing files? (Y/N) ")
  (DEFCONSTANT *MESS-QUERY-11*
     "Do yo wish to modify the rule-base interactively? (Y/N) ")

                    ; Prompt messages
                    ; ===============

  (DEFCONSTANT *MESS-PROMPT-1*
     "Enter the name of a text file (? for directory listing):  ")
  (DEFCONSTANT *MESS-PROMPT-2*
     "Press any key to continue...")
  (DEFCONSTANT *MESS-PROMPT-3*
     " cannot be opened.  Press any key to continue.")
  (DEFCONSTANT *MESS-PROMPT-4*
     "Enter name of file containing a rule base:  ")
  (DEFCONSTANT *MESS-PROMPT-5*
     "No rulebase exists.  Press any key to continue.")
  (DEFCONSTANT *MESS-PROMPT-6*
     "Enter file name for run (up to 8 letters):  ")
  (DEFCONSTANT *MESS-PROMPT-7*
     " already exists.  Press any key to try again.")
  (DEFCONSTANT *MESS-PROMPT-8*
     "Enter file name which contains the digrams:  ")
  (DEFCONSTANT *MESS-PROMPT-9*
     "Enter new rule and press return:  ")
  (DEFCONSTANT *MESS-PROMPT-10*
     "A(dd new rule/D(elete rule/R(eplace rule/V(iew rule /E(nviron/ESC to en
l: ")
  (DEFCONSTANT *MESS-PROMPT-11*
     "To move through rule base, use up/down arrows.  Esc to choose rule.")
  (DEFCONSTANT *MESS-PROMPT-12*
     "Press any key to interrupt scanning...")
  (DEFCONSTANT *MESS-PROMPT-13*
     "Current rule:  Press any key to return to edit mode...")
  (DEFCONSTANT *MESS-PROMPT-14*
     "Enter parameters for directory listing:  ")
  (DEFCONSTANT *MESS-PROMPT-15*
    "No files found - press any key to continue.")
  (DEFCONSTANT *MESS-PROMPT-16*
    "All files found - press any key to continue.")
  (DEFCONSTANT *MESS-PROMPT-17*
    "Press key to view next page.")

                    ; Information messages
                    ; ====================

  (DEFCONSTANT *MESS-INFO-1*
     " is being loaded into memory...")
  (DEFCONSTANT *MESS-INFO-2*
    "Please be patient while the scanner evaluator is loaded.")
```

```
                    ; Heading messages
                    ; ================

   (DEFCONSTANT *MESS-HEADING-1*
      "================================================================================
========")
   (DEFCONSTANT *MESS-HEADING-2*
      " A P R E S T O ")
   (DEFCONSTANT *MESS-HEADING-3*
      "A PRogrammable Evaluator for Scanning Text prOcessors")
   (DEFCONSTANT *MESS-HEADING-4*
      "Version 1.0")
   (DEFCONSTANT *MESS-HEADING-5*
      "Author: Annalu Waller, University of Cape Town")
   (DEFCONSTANT *MESS-HEADING-6*
      "                  31 July 1988")
   (DEFCONSTANT *MESS-HEADING-7*
      " END OF WRITER SYSTEM ")

 )

 (DEFUN DEFINE-PARAMETERS NIL

   ;; Define control parameters which are specific to this implementation:
   (DEFPARAMETER *GAPX*                6)          ; Column increment
   (DEFPARAMETER *GAPY*                2)          ; Row increment
   (DEFPARAMETER *ENV*               'ENV)         ; Environment variable

   ;; Read in date and set time:

   (DEFPARAMETER *DATE* (GET-DATE))
   (DEFVAR       *TIME* '(0 0)) )

 (DEFUN DEFINE-VARIABLES NIL

   ;; Define global variables to be used throughout the program:
   ;;          (may be changed during program)

   (DEFVAR *MAXEDITROW*                10)         ; Maximum rows edit window
   (DEFVAR *MAXEDITCOL*                79)         ; Maximum cols edit window
   (DEFVAR *XY-COORDS*             '((0 0)))       ; History of XY coordinates
   (DEFVAR *NEWTEXT*             '(*BLANK*))       ; List for processed text
   (DEFVAR *NEWWORD*                  NIL)         ; Current word
   (DEFVAR *LASTWORD*                 NIL)         ; Last word
   (DEFVAR *KEYSTROKES-YES*            0)          ; Affirmative keys counter
   (DEFVAR *CURRENT-YES*               0)          ;          Tempory counter
   (DEFVAR *KEYSTROKES-NO*             0)          ; Negative keys counter
   (DEFVAR *CURRENT-NO*                0)          ;          Tempory counter
   (DEFVAR *KEYSTROKES-DEL*            0)          ; Deletion counter
   (DEFVAR *TOTAL-ELEMENTS*            0)          ; Total element counter
   (DEFVAR *EFFICIENCY*                0)          ; Predicted efficiency
   (DEFVAR *DELAY*                     3)          ; Delay factor for scanner
   (DEFVAR *DEL-AV*                    0)          ; Deletion average
   (DEFVAR *OLD-DEL-AV*                0)          ; Deletion average history
   (DEFVAR *TEXT-FILE*               NIL)          ; Text file
   (DEFVAR *NEW-TEXT-FILE*           NIL)          ; File for processed text
   (DEFVAR *MIRRORED-TEXT-FILE*      NIL)          ; File for mirrored text
   (DEFVAR *STATS-FILE*              NIL)          ; File for statistics
```

```
  (DEFVAR *RULE-FILE*              NIL)          ; File for final rule base
  (DEFVAR *DIGRAM-NAME*            NIL)          ; Digram file name
  (DEFVAR *DIGRAM-FILE*            NIL)          ; Digram file
  (DEFVAR *SIMULATION-FILE*        NIL)          ; Text file for simulation
  (DEFVAR *SIMULATION-LIST*        NIL)          ; Current simulation item
  (DEFVAR *SIMULATION-RULES*       NIL)          ; Rules for simulation
  (DEFVAR *RULE-BASE*              NIL)          ; List of processing rules
  (DEFVAR *RULE-MODIFY*            NIL)          ; Flag
  (DEFVAR *RESTRICTIONS*           NIL)          ; Rules already tried
  (DEFVAR *QUIT*                   NIL)          ; Flag to signal end
  (DEFVAR *RUN-NAME*               NIL)          ; Run identification
  (DEFVAR *TECHNIQUE*              NIL)          ; Technique being used
  (DEFVAR *ENV-STORE*              NIL)          ; Store for environment
  (DEFVAR *GROUPS-NOT-WANTED*      NIL)          ; Environment history
  (DEFVAR *MULTIPLE-GROUPS*        NIL)          ; Multiple groups not used
  (DEFVAR *HIERARCHY*              NIL)          ; Group hierarchy
  (DEFVAR *HIERARCHY-LIST*         NIL)          ; Current hierarchy
  (DEFVAR *DIGRAM-LIST*            NIL)          ; List of digrams
  (DEFVAR *MAX-GROUPS*               1)          ; Number of groups in use
  (DEFVAR *PROCESSING-MODE*      'FREE))         ; Mode of text processing

(DEFUN DEFINE-ENVIRONMENT NIL

  (INIT-ENVIRONMENT)

  ;; Initialise attributes not handled above:
  (UPDATE-ENVIRONMENT :GROUP      NIL
                      :PREV-GROUP NIL
                      :STATUS     'PROCESSED)

  ;; Initialise edit window
  (SET-UP-GLOBAL-WINDOW)

  ;; Hide original cursor
  (POSITION-WINDOW-CURSOR -1 -1 *TERMINAL-IO*)

  ;; Initialise yes/no and yes/no/esc groups
  (MAKE-GROUP-VARIABLE 'YES-NO-GROUP `(("YES") ("NO")) 30 *SCREENLENGTH*)
  (MAKE-GROUP-VARIABLE 'YES-NO-ESC-GROUP `(("YES") ("NO") ("ESC"))
                                                    25 *SCREENLENGTH*)
  )

(DEFUN DEFINE-BASIC-RULES NIL

;;=============================================================================
;;  Set up basic rules
;;=============================================================================

  (REMEMBER *RULE-BASE*
            ((EQUAL (GET-ENVIRONMENT :ELEMENT) *QUITSYM*)
                    (SETQ *QUIT* T)
                    (UPDATE-ENVIRONMENT :STATUS 'PROCESSED)))
```

```
;;=============================================================================
;;  Set up basic simulation rules
;;=============================================================================

  (REMEMBER *SIMULATION-RULES*
      ((EQUAL ITEM *BLANK*)
                              (LIST *INSERTSYM*)))  ; If space => graphic

  (REMEMBER *SIMULATION-RULES*
      ((EQUAL ITEM (STRING *EOL*))
                              (LIST *RETURNSYM*))) ; If eol => return

  (REMEMBER *SIMULATION-RULES*
      ((INGROUP-P (OPPOSITE-CASE ITEM)
          (GET-GROUP (GET-ENVIRONMENT :GROUP)
                     :GROUP))                        ; If wrong case
                  (LIST *CARET* ITEM)))              ; Try different case
)


(DEFUN INIT-ENVIRONMENT NIL

  ;; Initialise environment group variable

  (UPDATE-ENVIRONMENT :RULE       NIL
                      :POSITION   0
                      :ELEMENT    NIL
                      :DESCRIPT   NIL)

  ;; Reset environment history to NIL

  (SETQ *GROUPS-NOT-WANTED* NIL))

(DEFUN DISPLAY-TITLE NIL                              ; Display headings
  (CLEAR-SCREEN *TERMINAL-IO*)                        ; Clear screen
  (DISPLAY-XY *MESS-HEADING-1* 0 0 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-1* 0 22 *TERMINAL-IO*)
  (DISPLAY-REVERSE-XY *MESS-HEADING-2* 30 7 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-3* 12 9 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-4* 31 11 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-5* 15 17 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-6* 17 19 *TERMINAL-IO*)
  (VIDEO-REVERSE *MESSAGE-WINDOW*)
  (DISPLAY-XY *MESS-PROMPT-2* 24 3 *MESSAGE-WINDOW*)
  (VIDEO-NORMAL *MESSAGE-WINDOW*)
  (READ-CHAR)                                         ; Wait for keypress
  (CLEAR-SCREEN *TERMINAL-IO*))                       ; Clear screen

(DEFUN DISPLAY-END NIL                                ; End off
  (DEFINE-MESSAGES)
  (CLEAR-SCREEN *TERMINAL-IO*)                        ; Clear screen
  (DISPLAY-XY *MESS-HEADING-1* 0 0 *TERMINAL-IO*)
  (DISPLAY-XY *MESS-HEADING-1* 0 22 *TERMINAL-IO*)
  (DISPLAY-REVERSE-XY *MESS-HEADING-7* 30 10 *TERMINAL-IO*)
  (SCREEN-BOTTOM *MESS-PROMPT-2*)
  (READ-CHAR)                                         ; Wait for keypress
  (CLEAR-SCREEN *TERMINAL-IO*))                       ; Clear screen
```

Appendix 6e - Program Listings

```
;;=============================================================================
;;FILE NAME:  READ
;;
;;  This group of functions define the input procedures used in the program
;;  environment.
;;=============================================================================

(DEFUN CHECK-KEYPRESS NIL
  (SEND *QUERY-IO* :CLEAR-INPUT)                    ; Clear the input buffer
  (DOTIMES (WAITT *DELAY*)
        (DOTIMES (WAIT *WAIT-DELAY*)))              ; Allow time for selection
  (COND ((SEND *QUERY-IO* :LISTEN) T)               ; Switch selection -> true
        (T NIL)))                                   ; Else NIL

(DEFUN SIMULATE (LIST)
  (COND ((NULL (INGROUP-P (STRING (CAR *SIMULATION-LIST*))
                          LIST))                    ; Not in list =>
         (INCF *KEYSTROKES-NO*)                     ;   Increment tally
         (INCF *CURRENT-NO*)                        ;   increment counter
         NIL)                                       ;   return NIL
        (T (INCF *KEYSTROKES-YES*)                  ; Else add to scan
           (INCF *CURRENT-YES*)                     ;   increment counter
           T)))                                     ; return true

(DEFUN CHECK-SPEED NIL
  (SETF *OLD-DEL-AV* *DEL-AV*)                      ; Remember current average
  (COND ((EQUAL *TOTAL-ELEMENTS* 0))                ; Calculate new average
        (T (SETQ *DEL-AV* (/ *KEYSTROKES-DEL*
                             *TOTAL-ELEMENTS*))))
  (COND ((> *DEL-AV* *OLD-DEL-AV*)                  ; If more errors =>
         (INCF *DELAY*))                            ;   slow down
        ((< *DEL-AV* *OLD-DEL-AV*)                  ; Else if less errors =>
         (COND ((AND (> *DELAY* *BASE-DELAY*)       ;   If delay above base
                     (NEQ *DELAY*                   ;     and not equal to
                          (+ *BASE-DELAY* 1)))      ;     increment =>
                (DECF *DELAY*))                     ;     speed up
               (T )))                               ;   Else do nothing
        (T )))                                      ; Else do nothing

(DEFUN KEYPRESSED (LIST)
  (COND ((EQUAL *PROCESSING-MODE* 'FREE)            ; Free text =>
    (COND ((CHECK-KEYPRESS)                         ;    Switch selected =>
                (INCF *KEYSTROKES-YES*)             ;       increment count
                (INCF *CURRENT-YES*)                ;       increment counter
                (CHECK-SPEED)                       ;       update speed
                T)                                  ;       return T
          (T (INCF *KEYSTROKES-NO*)                 ;    Else increment
                (INCF *CURRENT-NO*)                 ;       increment counter
                (CHECK-SPEED)                       ;       update speed
                NIL)))                              ;       return NIL
        (T (SIMULATE LIST))))                       ; Else simulate selection

(DEFUN CLEAR-INPUT (WINDOW)
  (SEND WINDOW :CLEAR-INPUT))                       ; Clear the input buffer

(DEFUN READ-FILE-CHAR (FILE)
  (SEND FILE :READ-CHAR-NO-HANG))                   ; Read character from file
```

Appendix 6e - Program Listings

```
(DEFUN UNREAD-FILE-CHAR (FILE CHAR)
  (SEND FILE :UNREAD-CHAR CHAR))                    ; Replace char in file

(DEFUN READ-FILE-LINE (FILE)
  (DO ((NEW-CHAR (READ-FILE-CHAR FILE)              ; Read in character and
                 (READ-FILE-CHAR FILE))             ;     repeat every round
       (LINE NIL))                                  ; Initialise variable
      ((EQUAL NEW-CHAR *EOL*) LINE)                 ; Return line at end
      (SETQ LINE (APPEND LINE                       ; Append new char on end
                         (LIST (STRING NEW-CHAR))))))
(DEFUN READ-A-CHAR (X Y)
  (GOTOXY X Y)                                      ; Position cursor
  (READ-CHAR))                                      ; Read a character
(DEFUN READ-A-SYMBOL (X Y)
  (GOTOXY X Y)                                      ; Position cursor
  (LET ((SYMBL (READ)))                             ; Read a sybol
      (COND ((EQUAL SYMBL ']) NIL)                  ;   Symbol empty -> NIL
            (T SYMBL))))                            ;   Else return the symbol

(DEFUN READ-LINE-SYMBOL (X Y WINDOW)
  (LET (RESULT ERR)                                  ; Read line but check
     (GOTOXY X Y WINDOW)
     (MULTIPLE-VALUE-SETQ (RESULT ERR)               ;   that result is valid
         (IGNORE-ERRORS
               (READ-FROM-STRING
                  (STRING-APPEND "(" (READ-LINE WINDOW) ")"))))
   RESULT))                                         ; Return result

(DEFUN READ-INTEGER (X Y)
  (LET ((NUMBER (READ-A-SYMBOL X Y)))               ; Read a string
     (COND ((NULL NUMBER) NIL)                      ; String empty ->return NIL
           ((NUMBERP NUMBER) NUMBER)                ; Else if number -> return
           (T (READ-INTEGER X Y)))))                ; Else attempt read again

(DEFUN READ-A-STRING (WORD GROUP WINDOW MARK)
  (COND ((EQUAL (CAR (LAST WORD)) MARK)             ; When mark is detected
                (FORGET-LAST WORD))                 ; -> return word w/o mark
    (T (SETQ TEMP (SCAN GROUP))                     ; Else scan group
       (COND ((NULL TEMP) (READ-A-STRING WORD       ; If nothing retry
                                GROUP WINDOW MARK))
             ((EQUAL TEMP *DELETESYM*)              ; If delete ->
               (POSITION-WINDOW-CURSOR              ;   move cursor left
                   (- (QUERY-WINDOW-X-POSITION WINDOW) 1)
                   (QUERY-WINDOW-Y-POSITION WINDOW)
                   WINDOW)
               (SEND WINDOW :DELETE-CHAR)           ; erase letter
               (READ-A-STRING (FORGET-LAST WORD)    ; Recall without last
                              GROUP WINDOW MARK))
             (T (SEND WINDOW :WRITE-STRING TEMP)    ; Else write letter
                (READ-A-STRING                      ; Recall write
                    (APPEND WORD (LIST TEMP))       ;    with new letter
                    GROUP WINDOW MARK))))))

(DEFUN READ-UNTIL-MARK (WORD GROUP WINDOW MARK)
  (DISPLAY-GROUP GROUP)                             ; Display group
  (LIST-TO-STRING                                   ; Convert a list of
     (READ-A-STRING WORD GROUP WINDOW MARK)))       ;  characters read in
```

Appendix 6f - Program Listings

```
;;=============================================================================
;;FILE NAME:  UTILities
;;
;;  This group of functions define the procedures used in more than one file,
;;  i.e. utilities used frequently.
;;=============================================================================

(DEFUN CHOP-OFF-LAST (OLD NEW)
  (COND ((EQUAL (LENGTH OLD) 1) NEW)                    ; Last element -> new
        (T (CHOP-OFF-LAST (CDR OLD)                     ; Else pass rest of string
             (APPEND NEW (LIST (CAR OLD)))))))          ; add first element to new

(DEFMACRO CLEAR-LIST (LISTVAR)
  `(SETQ ,LISTVAR NIL))                                 ; Reset variable

(DEFMACRO UPDATE-LIST (LISTVAR ADDITION)
  `(SETQ ,LISTVAR                                       ; Reset variable appending
     (APPEND ,LISTVAR
             (LIST ,ADDITION))))                        ; addition to end

(DEFMACRO TACK-ON-LIST (LISTVAR ADDITION)
  `(SETQ ,LISTVAR                                       ; Reset variable appending
     (APPEND (LIST ,ADDITION)
             ,LISTVAR)))                                ; addition to beginning

(DEFMACRO REMEMBER (STACK NEW)                          ; Add new phrase to stack
  `(COND ((MEMBER (QUOTE ,NEW)                          ; Is new addition member of
                  ,STACK :TEST 'EQUAL)                  ;    the stack
          NIL)                                          ; yes -> disregard
         (T (SETQ ,STACK                                ; no -> reset stack to
                  (APPEND ,STACK                        ;     the old stack with
                     (LIST (QUOTE ,NEW))))))))          ;     new addition appended

(DEFMACRO FORGET-LAST (STACK)                           ; To remove last phrase
  `(COND ((NULL ,STACK) NIL)                            ; Empty list -> NIL
         (T (SETQ ,STACK                                ; Else reset stack =
                  (CHOP-OFF-LAST ,STACK NIL)))))        ;  old less last

(DEFMACRO FORGET-FIRST (STACK)                          ; To remove first phrase
  `(COND ((NULL ,STACK) NIL)                            ; Empty list -> NIL
         (T (SETQ ,STACK (CDR ,STACK)))))               ; Else reset to tail

(DEFUN REMOVE-AT-POSITION (POSITION LIST)               ; To remove sublist
  (COND ((OR (< POSITION 0)                             ; If too low or
             (> POSITION (- (LENGTH LIST) 1)))          ;    too high =>
         LIST)                                          ;    the original list
        (T (DO ((POS 0 (INCF POS))                      ; Else go thru list
                (NEWLIST NIL)
                (CDRLIST LIST (CDR CDRLIST)))
               ((OR (NULL CDRLIST)                      ; until end or position
                    (EQUAL POS POSITION))
                (APPEND NEWLIST (CDR CDRLIST)))         ; => new list w/o sublist
             (SETQ NEWLIST                              ; New list = concatenation
        (COND ((NULL NEWLIST) (LIST (CAR CDRLIST)))
              (T (APPEND NEWLIST (LIST (CAR CDRLIST))))))))))
```

```
(DEFUN REMOVE-IF-EQUAL (ITEM CARSEQ CDRSEQ)          ; To remove sublist
  (COND ((NULL CDRSEQ) CARSEQ)                       ; If no tail => begining
        ((ATOM CDRSEQ)                               ; If tail not list =>
           (REMOVE-IF-EQUAL ITEM CARSEQ              ; Call again with
                            (LIST CDRSEQ)))          ;    list of tail
        ((EQUAL ITEM (CAR CDRSEQ))                   ; If match found =>
           (COND ((NULL CARSEQ) (CDR CDRSEQ))        ;   If no CAR => tail
                 (T (APPEND CARSEQ (CDR CDRSEQ)))))  ;   Else append w/o match
        ((NULL CARSEQ)                               ; If no CAR  =>
           (REMOVE-IF-EQUAL ITEM                     ; Call again with
                (LIST (CAR CDRSEQ)) (CDR CDRSEQ)))   ;    list of begining
        (T (REMOVE-IF-EQUAL ITEM                     ; Else call again with new
              (APPEND CARSEQ (LIST (CAR CDRSEQ)))    ;    begining and
              (CDR CDRSEQ)))))                       ;    new tail

(DEFUN ADD-LIST (SUBLIST POSITION SEQUENCE)          ; To add sublist to list
   (DO ((HIGHLEVEL (LENGTH SEQUENCE))                ; Note maximum position
        (NEWSEQ NIL)                                 ; Initialise new sequence
        (LEVEL 0 (INCF LEVEL))                       ; Counter
        (SEQ SEQUENCE (CDR SEQ)))                    ; Initialise sequence list
       ((COND ((EQUAL LEVEL POSITION) (SETQ NEWSEQ   ; If right level =>
                 (APPEND NEWSEQ
                         (LIST SUBLIST) SEQ)))       ;   append to middle
              ((> POSITION HIGHLEVEL)                ; If illegal position =>
                 (SETQ NEWSEQ SEQUENCE))             ;   old sequence
              (T NIL))
        NEWSEQ)
      (SETQ NEWSEQ
        (COND ((NULL NEWSEQ) (LIST (CAR SEQ)))       ; Begin sequence
              (T (APPEND NEWSEQ (LIST (CAR SEQ))))))))) ; or move on

(DEFUN COMPUTE-LEVEL (SUBLIST LIST)                  ; Find position of sublist
  (DO ((LEVEL 0 (INCF LEVEL))                        ; Initialise level
       (SEQUENCE LIST (CDR SEQUENCE)))               ; Go thru list
      ((COND ((NULL SEQUENCE) (SETQ LEVEL NIL) T)    ; If no  more => NIL
             ((EQUAL SUBLIST (CAR SEQUENCE)))        ; If equal => note position
             (T NIL))
       LEVEL)))                                      ; Return position

(DEFUN FIND-LIST-AT-LEVEL (LEVEL LIST)               ; Find sublist at position
  (DO ((LEV 0 (INCF LEV))                            ; Initialise counter
       (SEQUENCE LIST (CDR SEQUENCE)))               ; Go thru list
      ((COND ((NULL SEQUENCE) (SETQ SEQUENCE NIL)
                              T)                     ; If no  more => NIL
             ((EQUAL LEVEL LEV) T)                   ; If equal => end
             (T NIL))
       (CAR SEQUENCE))))                             ; Return sublist

(DEFUN SECOND-BUT-LAST (LIST)                        ; To assertain second last
  (COND ((< (LENGTH LIST) 2) *BLANK*)                ; If less than 2 => blank
        (T (NTH (- (LENGTH LIST) 2) LIST))))         ; Else second last item

(DEFMACRO MEMBER= (ITEM SET)                         ; Member test :TEST EQUAL
   `(MEMBER ,ITEM ,SET :TEST 'EQUAL))                ;   evaluate SET
```

```
(DEFUN INGROUP-P (ITEM LIST)
  (COND ((NULL LIST) NIL)                                ; Empty list -> return NIL
    (T (COND                                             ; Else
      ((EQUAL ITEM LIST))                                ;   Test basic equality,
      ((ATOM LIST) NIL)                                  ;   else Test if an atom
      ((MEMBER ITEM LIST :TEST 'EQUAL))                  ;   else Test single list
      ((MEMBER ITEM (CAR LIST) :TEST 'EQUAL))            ;   else Member CAR return
      (T (INGROUP-P ITEM (CDR LIST)))))))                ;   Else test rest of list

(DEFUN GROUP-NAME (CASE GROUP)                           ; Identify symbol name
  (FIND-SYMBOL (STRING-APPEND (STRING CASE)
                              (STRING GROUP))))

(DEFUN ROUND (NUMB)                                      ; Round to two decimals
  (/ (TRUNCATE (+ (* NUMB 100) 1)) 100))

(DEFUN ALIGN-NUMBER (NUMBER STREAM)                      ; Format numbers
  (COND ((< NUMBER 10  ) (PRINC "   " STREAM))          ; 1 digit  => 2 spaces
        ((< NUMBER 100 ) (PRINC "  "  STREAM))          ; 2 digits => 1 space
        (T ))                                            ; 3 digits => no space
  (PRINC NUMBER STREAM) )                                ; print the number

(DEFUN CONVERT-LIST-TO-STRING (LIST STRING)
  (COND ((NULL LIST) STRING)                             ; Empty list => string
        ((ATOM LIST) (STRING-APPEND STRING LIST))        ; Return string & last
        (T (CONVERT-LIST-TO-STRING (CDR LIST)            ; Recall convert
                        (STRING-APPEND STRING (CAR LIST))))))

(DEFUN ASCII (STRING)                                    ; Return ASCII number
  (COND ((NULL STRING) NIL)                              ; No string => NIL
        (T (CHAR STRING 0))))                            ; Return number

(DEFUN LIST-TO-STRING (LIST)
  (CONVERT-LIST-TO-STRING (CDR LIST) (CAR LIST)))

(DEFUN OPPOSITE-CASE (STRING)
  (COND ((NOT (ALPHA-CHAR-P (CHAR STRING 0))) NIL)  ; If not letter => NIL
        ((UPPER-CASE-P (CHAR STRING 0))             ; If capital =>
           (STRING (CHAR-DOWNCASE (CHAR STRING 0))));    lower case
        (T (STRING (CHAR-UPCASE (CHAR STRING 0))))));Else upper case

(DEFUN CLS NIL
  (CLEAR-SCREEN *TERMINAL-IO*))                          ; To clear display

;; Time and date functions

(DEFUN GET-TIME NIL
  (LET ((SEC NIL) (MIN NIL) (HR NIL)
        (DAY NIL) (MON NIL) (YR NIL))
    (MULTIPLE-VALUE-SETQ (SEC MIN HR DAY MON YR)
                         (GET-DECODED-TIME))
    (LIST HR MIN SEC)))
```

Appendix 6f - Program Listings

```
DEFUN GET-DATE NIL
  (LET ((SEC NIL) (MIN NIL) (HR NIL)
        (DAY NIL) (MON NIL) (YR NIL))
    (MULTIPLE-VALUE-SETQ (SEC MIN HR DAY MON YR)
                         (GET-DECODED-TIME))
    (LIST DAY (COND
          ((EQUAL MON 1) "JANUARY")     ((EQUAL MON 2) "FEBRUARY")
          ((EQUAL MON 3) "MARCH")       ((EQUAL MON 4) "APRIL")
          ((EQUAL MON 5) "MAY")         ((EQUAL MON 6) "JUNE")
          ((EQUAL MON 7) "JULY")        ((EQUAL MON 8) "AUGUST")
          ((EQUAL MON 9) "SEPTEMBER")   ((EQUAL MON 10) "OCTOBER")
          ((EQUAL MON 11) "NOVEMBER")   ((EQUAL MON 12) "DECEMBER")) YR)))

DEFUN TIME-ELAPSED (TIME1 TIME2)
  (LET ((HR  (- (CAR TIME1) (CAR TIME2)))
        (MIN (- (CADR TIME1) (CADR TIME2)))
        (SEC (- (CADDR TIME1) (CADDR TIME2))))
    (COND ((< SEC 0) (SETQ MIN (- MIN 1)) (SETQ SEC (+ SEC 60)))
          ((< MIN 0) (SETQ HR  (- HR  1)) (SETQ MIN (+ MIN 60)))
          (T ))
    (LIST HR MIN SEC)))
```

```
;;=============================================================================
;;FILE NAME:  FILEIO
;;  This group of functions simulate the scanning process by producing the
;;  equivalent text fed to it from a text file.
;;=============================================================================

(DEFUN FILE-ERROR (FILE-NAME MESSAGE)
  (SCREEN-BOTTOM (STRING-APPEND (STRING FILE-NAME)
                  MESSAGE))                          ; Display message
  (VIDEO-REVERSE *MESSAGE-WINDOW*)                   ; Switch highligting on
  (CLEAR-INPUT   *MESSAGE-WINDOW*)                   ; Clear input buffer
  (READ-CHAR     *MESSAGE-WINDOW*)                   ; Read a character
  (VIDEO-NORMAL  *MESSAGE-WINDOW*)                   ; Switch highligting off
  (REMOVE-BOTTOM))                                   ; Clean up

(DEFUN CHECK-DIR (WINDOW MESSAGE)
  (CLEAR-INPUT WINDOW)                               ; Clear input buffer
  (SCREEN-BOTTOM MESSAGE)                            ; Display message
  (VIDEO-REVERSE WINDOW)                             ; Switch highligting on
  (LET ((NAME (READ WINDOW)))                        ; Read a name
    (COND ((EQUAL NAME *QUESMARK*) (DIR))            ; If ?, get directory
          (T ))
    (VIDEO-NORMAL WINDOW)                            ; Switch highligting off
    (REMOVE-BOTTOM)                                  ; Clean up
   NAME))                                            ; Return name read

(DEFUN READ-TEXT-NAME (WINDOW)
  (LET ((NAME
          (CHECK-DIR WINDOW *MESS-PROMPT-1*)))       ; Read name
    (COND ((EQUAL NAME *QUESMARK*)                   ; If dir read, try again
                  (READ-TEXT-NAME WINDOW))
          (T NAME))))                                ; Else return name

(DEFUN SCAN-TEXT-NAME (WINDOW GROUP)
  (CLEAR-INPUT WINDOW)                               ; Clear input buffer
  (SCREEN-BOTTOM *MESS-PROMPT-1*)                    ; Display message
  (VIDEO-REVERSE WINDOW)                             ; Switch highligting on
  (LET ((NAME (READ-UNTIL-MARK NIL GROUP WINDOW      ; Scan for name
                               *RETURNSYM*)))
       (VIDEO-NORMAL WINDOW)                         ; Switch highligting off
       (REMOVE-BOTTOM)                               ; Clean up
       NAME))                                        ; Return name read

(DEFMACRO OPEN-OUTPUT-FILE (FILE-NAME MODE WINDOW &OPTIONAL GROUP)
  (LET ((TEMPF                                       ; Make note of file name
          (COND ((EQUAL ,MODE 'DIRECT)               ; read in either
                 (READ-TEXT-NAME ,WINDOW))           ; by direct reading
                (T (SCAN-TEXT-NAME ,WINDOW
                                   ,GROUP)))))       ; or by scanning
        (COND ((NULL (PROBE-FILE
                       (PATHNAME TEMPF)))            ; If file does not exist
               (SETF ,FILE-NAME                      ; Give stream a name
                     (OPEN (PATHNAME TEMPF)          ; open a file
                           :DIRECTION :OUTPUT)))     ; for output
              (T (FILE-ERROR TEMPF
                             *MESS-PROMPT-7*)        ; Display message
                 NIL))))                             ; return nil
```

Appendix 3g -Program Listings

```
(DEFMACRO OPEN-PRINTER (FILE-NAME)
  (COND ((NULL (PROBE-FILE
                (MAKE-PATHNAME :DEVICE 'PRN))) ; If printer is not available
         (FILE-ERROR "PRINTER"
                     *MESS-PROMPT-3*)          ;      report error
         NIL)                                  ;      return NIL
        (T (SETF ,FILE-NAME                    ; Give stream a name
           (OPEN (MAKE-PATHNAME :DEVICE 'PRN) ; open the printer
            :DIRECTION :OUTPUT)))))            ; for output

(DEFMACRO OPEN-INPUT-FILE (FILE-NAME MODE WINDOW &OPTIONAL GROUP)
  (LET ((TEMPF                                 ; Make note of file name
         (COND ((EQUAL ,MODE 'DIRECT)          ; read in either
                (READ-TEXT-NAME ,WINDOW))      ; by direct reading
               (T (SCAN-TEXT-NAME ,WINDOW
                                  ,GROUP))))) ; or by scanning
        (COND ((NULL (PROBE-FILE
                      (PATHNAME TEMPF)))       ; If file does not exist
               (FILE-ERROR TEMPF
                           *MESS-PROMPT-3*)     ;       report error
               NIL)                            ;       return NIL
              (T (SETF ,FILE-NAME              ; Else give stream a name
                       (OPEN (PATHNAME TEMPF)  ; open a file
                         :DIRECTION :INPUT))))) ; for input
        TEMPF))                                ; return name

(DEFUN PRINT-DIR (DIR-LIST LINE)
  (COND ((NULL DIR-LIST) )                              ; End of list
        ((> LINE 18) (SCREEN-BOTTOM *MESS-PROMPT-17*); print message
                     (READ-CHAR *MESSAGE-WINDOW*)     ; wait
                     (CLEAR-SCREEN *TERMINAL-IO*)     ; clear screen
                     (PRINT-DIR DIR-LIST 0))          ; print rest
        (T (PRINT (CAR DIR-LIST))                     ; print file
           (PRINT-DIR (CDR DIR-LIST) (+ LINE 1)))))   ; Print rest

(DEFUN DIR NIL
  (LET* ((SCRMESS (SCREEN-BOTTOM *MESS-PROMPT-14*))  ; Display message
         (PATHN   (READ-LINE *MESSAGE-WINDOW*))      ; Read parameter
         (DIREC   (DIRECTORY PATHN)))                ; get the directory
    (COND ((NULL DIREC) (SCREEN-BOTTOM *MESS-PROMPT-15*)
                        (READ-CHAR *MESSAGE-WINDOW*))
          (T (CLEAR-SCREEN *TERMINAL-IO*)
             (PRINT-DIR DIREC 0)                      ; Print directory
             (SCREEN-BOTTOM *MESS-PROMPT-16*)         ; Wait for input
             (READ-CHAR *MESSAGE-WINDOW*)
             (CLEAR-SCREEN *TERMINAL-IO*))))))
```

```
;;=============================================================================
;;FILE NAME:  ACCESS
;;
;;  This group of functions define and access the data structures fundamental
;;  to this programming environment.
;;=============================================================================

;;=============================================================================
;;  SECTION I:  Identify window for text processing and messaging.
;;=============================================================================

(DEFUN SET-UP-GLOBAL-WINDOW NIL

;; Set up editing window:

  (SETF *EDIT-WINDOW*                                    ; Give window value by:
    (MAKE-WINDOW-STREAM                                  ; Setting up window stream:
            :HEIGHT *MAXEDITROW*                         ;   give maximum rows
            :WIDTH  *MAXEDITCOL*))                       ;   give maximum columns

;; Set up rule editing window:

  (SETF *RULE-WINDOW*                                    ; Give window value by:
    (MAKE-WINDOW-STREAM                                  ; Setting up window stream:
            :TOP (- *SCREENLENGTH* *MAXRULEROW*)         ;   where to start window
            :HEIGHT *MAXRULEROW*                         ;   give maximum rows
            :WIDTH  *MAXRULECOL*))                       ;   give maximum columns

;; Set up messaging window:

  (SETF *MESSAGE-WINDOW*                                 ; Give window value by:
    (MAKE-WINDOW-STREAM                                  ; Setting up window stream:
           :TOP (- *SCREENLENGTH* *MAXMESSAGEROW*)       ;   where to start window
           :HEIGHT *MAXMESSAGEROW*                       ;   give maximum rows
           :WIDTH  *MAXMESSAGECOL*)))                    ;   give maximum columns

;;=============================================================================
;;  SECTION II:  Group variables with attributes.
;;=============================================================================

(DEFUN REFORMAT-TO-ROWS (COLUMN-LIST ROW-LIST)

;; This function reformats a one column list into rows.

  (COND ((NULL COLUMN-LIST) ROW-LIST)                    ; End when no more list
        ((NULL ROW-LIST)                                 ; If beginning =>
           (REFORMAT-TO-ROWS (CDR COLUMN-LIST)           ;   list first item
                             (LIST (LIST (CAR COLUMN-LIST)))))
        (T (REFORMAT-TO-ROWS (CDR COLUMN-LIST)           ; Else add
                (APPEND ROW-LIST (LIST (LIST (CAR COLUMN-LIST))))))))

(DEFUN CHECK-FORMAT (ITEM-LIST)

 ; This function checks that if there is only one column, the item list
 ; is reformatted to reflect single row colums.
```

```
  (COND ((EQUAL (LENGTH ITEM-LIST) 1)                    ; Length = 1 =>
         (REFORMAT-TO-ROWS (CAR ITEM-LIST) NIL))         ; Reformat
        (T ITEM-LIST)))                                  ; Else return list

(DEFUN UPDATE-GROUP (NAME &REST OPTIONS)

;; This function updates a group variable using the following attributes:
;;    NAME         - compulsory name identifies specific group variable
;;    :GROUP       - elements which constitute the group variable
;;    :STARTX    - position of top left X-coordinate on screen
;;    :STARTY    - position of top left Y-coordinate on screen

  (DO ((X OPTIONS (CDDR X)))                             ; Set loop to go thru list
    ((NULL X))                                           ; End when list is complete
    (CASE (CAR X)                                        ; Identify attribute used

;; Set group value (i.e. all elements belonging to group in order):
   (:GROUP  (SETF (GET NAME 'GROUP)  (CHECK-FORMAT (CADR X))))

;; Set starting X and Y coordinates:
   (:STARTX (SETF (GET NAME 'STARTX) (CADR X)))
   (:STARTY (SETF (GET NAME 'STARTY) (CADR X)))))

;; If the X or Y starting coordinates have changed, the window attributes
;; must be recconfigured:
  (COND ((OR (MEMBER :STARTX OPTIONS)
             (MEMBER :STARTY OPTIONS))
     (SETF (GET NAME 'WINDOW)                            ; Give window value by:
       (MAKE-WINDOW-STREAM                              ;  Setting up window stream
         :HEIGHT (+ (- (GET-GROUP NAME :ENDY)            ;    height 4 rows extra
                       (GET-GROUP NAME :STARTY))
                    4)
         :WIDTH  (+ (- (GET-GROUP NAME :ENDX)            ;   give width 4 columns "
                       (GET-GROUP NAME :STARTX))
                    8)
         :TOP    (- (GET-GROUP NAME :STARTY) 2)        ;   start window 2 rows up
         :LEFT   (- (GET-GROUP NAME :STARTX) 2)        ;     "       "  2 cols left
         :CURSORPOS-X 2                                ;   initial X coordinate
         :CURSORPOS-Y 2))                              ;   initial Y coordinate
    (SETF (GET NAME 'STARTX) 2)                        ;   Reset starting X coord
    (SETF (GET NAME 'STARTY) 2))                       ;   Reset starting Y coord
       (T NIL))                                        ; Else do nothing
  NAME)                                                ; Return name
```

```
(DEFUN GET-GROUP (NAME ATTRIBUTE &OPTIONAL ROW COL)
  (CASE ATTRIBUTE
    (:GROUP    (GET NAME 'GROUP))                    ; Identify group matrix
    (:STARTX   (GET NAME 'STARTX))                   ; Identify starting X coord
    (:STARTY   (GET NAME 'STARTY))                   ; Identify starting Y coord
    (:COL   (MAXCOL (GET-GROUP NAME :GROUP)))        ; Calculate number columns
    (:ROW   (MAXROW (GET-GROUP NAME :GROUP)))        ; Calculate number rows
    (:ENDX     (+ (GET-GROUP NAME :STARTX)           ; Compute ending X value
               (* (- (GET-GROUP NAME :COL) 1)
                  *GAPX*)))
    (:ENDY     (+ (GET-GROUP NAME :STARTY)           ; Compute ending Y value
               (* (- (GET-GROUP NAME :ROW) 1)
                  *GAPY*)))
    (:WINDOW   (GET NAME 'WINDOW))                   ; Identify window stream
    (:ELEMENT  (COND ((AND (>= ROW 0)                ; Within bounds -> carry on
                           (<= ROW (GET-GROUP NAME :ROW))
                           (>= COL 0)
                           (<= COL (GET-GROUP NAME :COL)))
      (COND
  ((EQUAL ROW 0) (CAR         (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 1) (CADR         (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 2) (CADDR         (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 3) (CADDR (CDR     (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 4) (CADDR (CDDR      (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 5) (CADDR (CDDDR      (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 6) (CADDR (CDDDR (CDR (NTH COL (GET-GROUP NAME :GROUP))))))
  ((EQUAL ROW 7) (CADDR (CDDDR (CDDR (NTH COL (GET-GROUP NAME :GROUP))))))
  ((EQUAL ROW 8) (CADDR (CDDDR (CDDDR (NTH COL (GET-GROUP NAME :GROUP)))))))
                     (T 'OUT-OF-BOUNDS)))))        ; Else out of bounds

(DEFUN MODIFY-AN-ELEMENT (NAME ROW COL ELEMENT)
  (COND ((AND (>= ROW 0)                             ; Within bounds -> carry on
              (<  ROW (GET-GROUP NAME :ROW))
              (>= COL 0)
              (<  COL (GET-GROUP NAME :COL)))
         (RPLACA
                (COND                                ; Replace relevant CAR with
  ((EQUAL ROW 0) (             NTH COL (GET-GROUP NAME :GROUP)))
  ((EQUAL ROW 1) (CDR          (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 2) (CDDR          (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 3) (CDDDR          (NTH COL (GET-GROUP NAME :GROUP))))
  ((EQUAL ROW 4) (CDDDR (CDR      (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 5) (CDDDR (CDDR      (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 6) (CDDDR (CDDDR      (NTH COL (GET-GROUP NAME :GROUP)))))
  ((EQUAL ROW 7) (CDDDR (CDDDR (CDR (NTH COL (GET-GROUP NAME :GROUP))))))
  ((EQUAL ROW 8) (CDDDR (CDDDR (CDDR (NTH COL (GET-GROUP NAME :GROUP)))))))
                 ELEMENT))                           ; Replacement value
        (T 'OUT-OF-BOUNDS)))                         ; Else out of bounds

(DEFUN MAKE-GROUP-VARIABLE (NAME GROUP X Y)
  (UPDATE-GROUP NAME                                 ; Set up group variable by:
    :GROUP  GROUP                                    ; Set up group
    :STARTX X                                        ; Starting X coord value
    :STARTY Y))                                      ; Starting Y coord value
```

```
Appendix 6h - Program Listings
(DEFUN MAXCOL (GROUP)
  (LENGTH GROUP))                                    ; Length of matrix list

(DEFUN MAXROW (GROUP)
  (COND ((NULL GROUP) 0)                             ; If empty -> NIL
        (T (FIND-MAXROW GROUP 0))))                  ; Else find the maximum

(DEFUN FIND-MAXROW (GROUP MAXIMUM)
  (COND ((NULL GROUP) MAXIMUM)                       ; End of list -> return max
        (T (FIND-MAXROW (CDR GROUP)                  ; Else find maximum of rest
           (MAX (LENGTH (CAR GROUP))                 ;   by finding column max
                MAXIMUM)))))                         ;   and old maximum

;;=============================================================================
;;  SECTION III:  Functions which counter unecessary group repetitions.
;;=============================================================================

(DEFUN REMEMBER-GROUP NIL

;; If multiple-groups, then a record of groups scanned unsuccessfully is kept:

  (COND (*MULTIPLE-GROUPS*                           ; If multiple => go ahead
    (COND ((EQUAL (LENGTH *GROUPS-NOT-WANTED*)
                  *MAX-GROUPS*)                      ; If maximum tried =>
              (CLEAR-LIST *GROUPS-NOT-WANTED*))      ;    clear record
          (T ))                                      ; Else nothing
    (UPDATE-LIST *GROUPS-NOT-WANTED*                 ; Add current group
              (GET-ENVIRONMENT :NAME)))
       (T NIL)))                                     ; Else if single => nothing

(DEFUN FIND-POSSIBLE-GROUP (GROUP-LIST)

;; Identify next group in default order:

  (COND ((NULL *GROUPS-NOT-WANTED*)                  ; No restrictions => leave
         (CAR GROUP-LIST))                           ;       return first group
        ((NULL GROUP-LIST)                           ; No choice =>
         (CAR *GROUPS-NOT-WANTED*))                  ;       return first group
        ((MEMBER= (CAR GROUP-LIST)
                  *GROUPS-NOT-WANTED*)               ; Restricted => try next
         (FIND-POSSIBLE-GROUP (CDR GROUP-LIST)))
        (T (CAR GROUP-LIST))))                       ;      use first group

(DEFUN GROUP-ATTEMPTED-P (NEW-GROUP)

 ; Has the group been scanned before?

  (COND ((NULL NEW-GROUP) NIL)                       ; Nil group => NIL
        ((MEMBER= NEW-GROUP *GROUPS-NOT-WANTED*)     ; If group has been scanned
          T)                                         ;  & return TRUE
        (T NIL)))                                    ; Else nil
```

```
;;=============================================================================
;;  SECTION IV:  Environment structure with attributes.
;;=============================================================================

(DEFUN RECORD-ENVIRONMENT NIL

;; Make a copy of the current environment:

  (SETQ *ENV-STORE* (NOTE-ENVIRONMENT)))

(DEFUN RECALL-ENVIRONMENT (OLD-ENV ATTRIBUTE)

;; To reinstate an old environment attribute:

  (EVAL `(UPDATE-ENVIRONMENT ,ATTRIBUTE
           (FIND-ATTRIBUTE ,ATTRIBUTE (QUOTE ,OLD-ENV))
           :REINSTATE)))

(DEFUN REINSTATE-ENVIRONMENT (OLD-ENV)

;; To reinstate an entire environment:

  (DO ((X OLD-ENV (CDDR X)))                          ; Set loop to go thru list
    ((NULL X))                                        ; End when list is complete
    (COND ((EQUAL (CAR X) ':ELEMENT))                 ; Ignore :ELEMENT reinstate
          (T (RECALL-ENVIRONMENT X (CAR X)))))        ; Reinstate first in list

  (UPDATE-ENVIRONMENT :CONTINUE :REINSTATE))          ; Continue with rule tests

(DEFUN RESET-CONTINUE NIL
  (SETF (GET *ENV* 'CONTINUE) NIL))                   ; Set continuation flag off

(DEFUN UPDATE-ENVIRONMENT (&REST OPTIONS)

;; This function updates the environment using the following attributes:
;;    :GROUP      - name of group variable
;;    ;CASE       - case of group
;;    :RULE       - rule used
;;    :STATUS     - flags whether element has been processed or not
;;    :CONTINUE   - flags whether rule-base must be searched further or not
;;    :POSITION   - position within word or sentence structure
;;    :ELEMENT    - last element chosen (must be a character/string)
;;    :DESCRIPT   - describes linguistic attribute
;; Always see that continuation flag is off:

  (RESET-CONTINUE)

;; If a group has been updated, or if a group has been ignored =>
;; test if group has already been attempted, and if so =>
;; reinstate old environment:

    (COND ((AND (NOT (INGROUP-P ':REINSTATE OPTIONS))
                (OR (INGROUP-P ':GROUP OPTIONS)
                    (EQUAL (GET-ENVIRONMENT :ELEMENT) *NULSYM*)))
           (COND ((GROUP-ATTEMPTED-P
                        (FIND-ATTRIBUTE ':GROUP OPTIONS))
```

```
 ;; (GET-ENVIRONMENT :NAME))
                        (SETF (GET *ENV* 'CONTINUE) T))
                    (T NIL)))
          (T NIL) )

;; If continue has been set end, els carry on with updates:

  (COND ((GET-ENVIRONMENT :CONTINUE) NIL)
        (T

;; Carry on with updates:

  (DO ((X OPTIONS (CDDR X)))                          ; Set loop to go thru list
    ((NULL X))                                        ; End when list is complete
    (CASE (CAR X)                                     ; Identify attribute used

;; Update and display new group (clear the window first):
       (:GROUP    (SETF (GET *ENV* 'GROUP)    (CADR X))
                  (CLEAR-SCREEN
                       (GET-GROUP (GROUP-NAME (GET *ENV* 'CASE) (CADR X))
                                  :WINDOW))
                  (DISPLAY-GROUP  (GROUP-NAME (GET *ENV* 'CASE) (CADR X))))

;; Update case and display new group:
       (:CASE     (SETF (GET *ENV* 'CASE)      (CADR X))
                  (DISPLAY-GROUP (GROUP-NAME (CADR X)
                                             (GET *ENV* 'GROUP))))

;; If rule not NIL, add it to list, else reset rule to NIL:
       (:RULE     (COND ((NULL (CADR X)) (SETF (GET *ENV* 'RULE) NIL))
                        (T (SETF (GET *ENV* 'RULE)
                                 (APPEND (GET-ENVIRONMENT :RULE)
                                         (LIST (CADR X))))))))

;; Set status:
       (:STATUS   (SETF (GET *ENV* 'STATUS)   (CADR X)))

;; Flag continuation:
       (:CONTINUE (SETF (GET *ENV* 'CONTINUE) T))

;; Set character position:
       (:POSITION (SETF (GET *ENV* 'POSITION) (CADR X)))

;; If no element was passed, use a blank, else use new value to reset element:
       (:ELEMENT  (COND ((NULL (CADR X)) (SETF (GET *ENV* 'ELEMENT) *NULSYM*))
                        (T (SETF (GET *ENV* 'ELEMENT) (CADR X))
                           (UPDATE-ENVIRONMENT :STATUS 'NOT-PROCESSED))))

;; Set description of environment:
       (:DESCRIPT (SETF (GET *ENV* 'DESCRIPT) (CADR X)))
    ))                                                ; End of DO
  ))                                                  ; End of COND
)                                                     ; End of UPDATE
```

```
(DEFUN GET-ENVIRONMENT (ATTRIBUTE)
  (CASE ATTRIBUTE
    (:GROUP    (GROUP-NAME (GET *ENV* 'CASE)
                           (GET *ENV* 'GROUP)))     ; Identify name of group
    (:NAME      (GET *ENV* 'GROUP))                 ; Identify collective name
    (:CASE      (GET *ENV* 'CASE))                  ; Identify group case
    (:RULE      (GET *ENV* 'RULE))                  ; Identify rule
    (:STATUS    (GET *ENV* 'STATUS))                ; Identify status
    (:CONTINUE  (GET *ENV* 'CONTINUE))              ; Identify continue flag
    (:POSITION  (GET *ENV* 'POSITION))              ; Identify position
    (:ELEMENT   (GET *ENV* 'ELEMENT))               ; Identify element
    (:DESCRIPT  (GET *ENV* 'DESCRIPT))))            ; Identify description

(DEFUN GET-LAST-ENVIRONMENT (ATTRIBUTE)
  (FIND-ATTRIBUTE ATTRIBUTE *ENV-STORE*))           ; Find last entry attribute

(DEFUN FIND-ATTRIBUTE (ATTRIBUTE LIST)
  (COND ((NULL LIST) NIL)                           ; Check list not empty
        ((NULL ATTRIBUTE) NIL)                      ; Check attribute not empty
        ((EQUAL ATTRIBUTE (CAR LIST))               ; If attribute found =>
           (CADR LIST))                             ;   return second element
        (T (FIND-ATTRIBUTE ATTRIBUTE (CDDR LIST))))) ; Else scan rest of list

(DEFUN NOTE-ENVIRONMENT NIL                         ; Note the environment
  `(:GROUP ,(GET-ENVIRONMENT :NAME)                 ;        collective name
    :CASE ,(GET-ENVIRONMENT :CASE)                  ;        case
    :RULE ,(GET-ENVIRONMENT :RULE)                  ;        rule
    :STATUS ,(GET-ENVIRONMENT :STATUS)              ;        status
    :CONTINUE ,(GET-ENVIRONMENT :CONTINUE)          ;        continue
    :POSITION ,(GET-ENVIRONMENT :POSITION)          ;        position
    :ELEMENT ,(GET-ENVIRONMENT :ELEMENT)            ;        last character
    :DESCRIPT ,(GET-ENVIRONMENT :DESCRIPT)))        ;        description
```

```
;;=============================================================================
;;FILE NAME:  CURSOR
;;
;;  This group of functions define the procedures used to control cursor
;;  movement on the screen.
;;=============================================================================

;;-----------------------------------------------------------------------------
;;  The following coordinate functions record and interrogate the position
;;  of the cursor in a specified window.
;;-----------------------------------------------------------------------------

(DEFUN QUERY-WINDOW-POSITION (WINDOW)
  (MULTIPLE-VALUE-SETQ (X Y)                          ; Give new values by:
     (SEND WINDOW :CURSORPOS))                        ; Recording current coords
  (LIST X Y))                                         ; Return list of X and Y

(DEFUN QUERY-WINDOW-X-POSITION (WINDOW)
  (CAR (QUERY-WINDOW-POSITION WINDOW)))               ; Return X coordinate

(DEFUN QUERY-WINDOW-Y-POSITION (WINDOW)
  (CADR (QUERY-WINDOW-POSITION WINDOW)))              ; Return Y coordinate

(DEFUN POSITION-WINDOW-CURSOR (X Y WINDOW)
  (SEND WINDOW :SET-CURSORPOS X Y))                   ; Position cursor

;;-----------------------------------------------------------------------------
;;  The following coordinate functions record and interrogate the position
;;  of the cursor in the edit window.
;;-----------------------------------------------------------------------------

(DEFUN RECORD-XY-POSITION (X &REST Y)
  (COND ((> (LENGTH *XY-COORDS*) *MAXEDITCOL*)        ; Check length
         (FORGET-FIRST *XY-COORDS*))
        (T ))
  (SETF *XY-COORDS*                                   ; Give new value by:
    (APPEND *XY-COORDS* (LIST                         ; Append new XY coord list:
        (COND ((LISTP X) X)                           ;   If X is a list -> use X
              (T (APPEND (LIST X) Y)))))))            ;   Else list X and Y

(DEFUN UNDO-XY-POSITION NIL
  (FORGET-LAST *XY-COORDS*))                          ; Remove last coordinate

(DEFUN REMEMBER-XY-POSITION NIL
  (CAR (LAST *XY-COORDS*)))                           ; Identify last XY position

(DEFUN CURRENT-X NIL
  (CAR (REMEMBER-XY-POSITION)))                       ; Identify last X position

(DEFUN CURRENT-Y NIL
  (CADR (REMEMBER-XY-POSITION)))                      ; Identify last Y position

(DEFUN REVERSE-XY-POSITION (LENGTH)
  (COND ((EQUAL (LENGTH *XY-COORDS*) 1))              ; beginning -> do nothing
        (T (DOTIMES (LOOP LENGTH)                     ; Else set loop counter
             (UNDO-XY-POSITION)))))                   ;      remove last coords
```

```
;;-------------------------------------------------------------------------
;;  The following functions positions cursor in a specified window.
;;-------------------------------------------------------------------------

(DEFUN DELETE-LAST-CHAR (WINDOW LENGTH)
  (DOTIMES (LOOP LENGTH)                              ; Beginning -> do nothing
    (REVERSE-XY-POSITION 1)                           ; reverse X, Y coords
    (POSITION-WINDOW-CURSOR                           ; Position cursor
                (CURRENT-X) (CURRENT-Y) WINDOW)
    (SEND WINDOW :DELETE-CHAR))                       ; Delete at cursor position
  (POSITION-CURSOR (CURRENT-X) (CURRENT-Y)            ; Position highlighted
                   WINDOW))                           ;   cursor

(DEFUN ADVANCE-LINE NIL
  (RECORD-XY-POSITION (+ (CURRENT-X) 1)               ; Move across to accomodate
                      (CURRENT-Y))                    ; deletion of return
  (RECORD-XY-POSITION 0                               ; Reset column
                      (+ (CURRENT-Y) 1)))             ; Move down 1 row

(DEFUN ADVANCE-XY-POSITION (DISTANCE)
  (DOTIMES (COUNT DISTANCE)                           ; Initiate counter
    (COND ((EQUAL (CURRENT-X) (- *MAXEDITCOL* 1))     ; If line full ->
           (ADVANCE-LINE))                            ;    move row down 1
          (T (RECORD-XY-POSITION                      ; Else
               (+ (CURRENT-X) 1) (CURRENT-Y)))))))    ;    move column across

(DEFUN POSITION-CURSOR (X Y WINDOW)
  (SETQ TMP-STORE (QUERY-WINDOW-POSITION WINDOW))     ; Memorise cursor position
  (DISPLAY-XY *CURSOR* X Y WINDOW)                    ; Set cursor
  (POSITION-WINDOW-CURSOR (CAR TMP-STORE) (CADR TMP-STORE)
                          WINDOW))                    ; Reposition cursor

(DEFUN REMOVE-CURSOR (X Y)
  (DISPLAY-XY *BLANK* X Y *EDIT-WINDOW*))                  ; Set cursor

(DEFUN MOVE-ON (DISTANCE)
  (POSITION-CURSOR (CURRENT-X) (CURRENT-Y)            ; Show cursor
                   *EDIT-WINDOW*)                     ; using the edit window
  (ADVANCE-XY-POSITION DISTANCE))                     ; Move XY
        position on
```

```
;;=============================================================================
;;FILE NAME:  RUNOUTput
;;  This file contains the functions which store the text which is created by
;;  the scanner.  The final rule base is also stored.
;;=============================================================================

(DEFUN OPEN-ALL-RUN-FILES (FILE-NAME)
  (SETF *NEW-TEXT-FILE*                          ; File for created text
        (OPEN (PATHNAME (STRING-APPEND FILE-NAME ".NEW"))
              :DIRECTION :OUTPUT))
  (SETF *MIRRORED-TEXT-FILE*                     ; File for selection
        (OPEN (PATHNAME (STRING-APPEND FILE-NAME ".MIR"))
              :DIRECTION :OUTPUT))
  (SETF *STATS-FILE*                             ; File for stats
        (OPEN (PATHNAME (STRING-APPEND FILE-NAME ".STA"))
              :DIRECTION :OUTPUT)))

(DEFUN CLOSE-RUN-FILES NIL
  (CLOSE-ALL-FILES))

(DEFUN STORE-RULES NIL
  (SETF *RULE-FILE*                              ; File for rules
        (OPEN (PATHNAME (STRING-APPEND *RUN-NAME* ".RUL"))
              :DIRECTION :OUTPUT))

  (FORMAT *RULE-FILE* *MESS-STATS-30* *RUN-NAME* *DATE*)
  (FORMAT *RULE-FILE* *MESS-STATS-31* *TECHNIQUE*)
  (FORMAT *RULE-FILE* *MESS-STATS-32* *MODIFY-RULE* *MULTIPLE-GROUPS*)
  (FORMAT *RULE-FILE* *MESS-STATS-33* *MAXEDITROW*)
  (FORMAT *RULE-FILE* *MESS-STATS-34* *BASE-DELAY*)
  (FORMAT *RULE-FILE* *MESS-STATS-35* *MAX-GROUPS*)

  (FORMAT *RULE-FILE* *MESS-STATS-37* *HIERARCHY*)

  (DO ((LST *RULE-BASE* (CDR LST)))
      ((NULL LST) (CLOSE *RULE-FILE*))
    (FORMAT *RULE-FILE* *MESS-STATS-38* (CAAR LST) (CADAR LST))))

(DEFUN ACCEPT-EXISTING-NAMES (NAME)
  (SCREEN-BOTTOM (STRING-APPEND (STRING NAME)      ; Offer escape
                                *MESS-QUERY-10*))
  (YES-NO-P))                                      ; Return true of nil

(DEFUN CHECK-RUN-FILE (FILE-NAME)
  (COND ((NULL (PROBE-FILE (PATHNAME FILE-NAME)))   ; If file does not exist
         (OPEN-ALL-RUN-FILES FILE-NAME) T)          ;    then open files
        (T (COND ((ACCEPT-EXISTING-NAMES FILE-NAME) ; Else rewrite?
                  (OPEN-ALL-RUN-FILES FILE-NAME)    ;      open files
                  T)                                ;      return T
                 (T NIL)))))                        ; Else return NIL

(DEFUN OPEN-RUN-FILES NIL
  (SETQ *RUN-NAME* (CHECK-DIR *MESSAGE-WINDOW*
                              *MESS-PROMPT-6*))     ; Read a name
  (COND ((CHECK-RUN-FILE *RUN-NAME*))               ; Run name is valid
        (T (OPEN-RUN-FILES)))                       ; Else retry
  (REMOVE-BOTTOM))                                  ; Clean up
```

```
;;=============================================================================
;;FILE NAME:  SCREEN
;;  This group of functions define the procedures used to control screen
;;  printing.
;;=============================================================================

(DEFUN VIDEO-REVERSE (WINDOW)
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :SET-ATTRIBUTE #X70))))      ; Else reverse video

(DEFUN VIDEO-NORMAL (WINDOW)
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :SET-ATTRIBUTE #X07))))      ; Else normal video

(DEFUN CLEAR-SCREEN (WINDOW)                         ; To clear screen
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :CLEAR-SCREEN))))            ; Else clear window

(DEFUN CLEAR-EOL (WINDOW)                            ; To clear till end of line
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :CLEAR-EOL))))

(DEFUN CLEAR-EOS (WINDOW)                            ; Clear till end of screen
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :CLEAR-EOS))))

(DEFUN GOTOXY (X Y WINDOW)                           ; Goto coordinates X and Y
  (COND ((NULL WINDOW) NIL)                          ; Check non null
        (T (SEND WINDOW :SET-CURSORPOS X Y))))

(DEFUN DISPLAY-XY (STRNG X Y WINDOW)                 ; Print string at XY coords
  (GOTOXY X Y WINDOW)                                ; Set cursor at X Y coords
  (FORMAT WINDOW STRNG))                             ; Print string

(DEFUN DISPLAY-REVERSE-XY (STRNG X Y WINDOW)
  (VIDEO-REVERSE WINDOW)                             ; Switch highlighting on
  (DISPLAY-XY STRNG X Y WINDOW)                      ; Print string at XYcoords
  (VIDEO-NORMAL WINDOW))                             ; Switch highlighting off

(DEFUN DISPLAY-LIST (LIST)                           ; Print list on screen
  (COND ((NULL LIST))                                ; End of list -> return NIL
        (T (PRINC (CAR LIST))                        ;   print first element
           (DISPLAY-LIST (CDR LIST)))))              ;   print rest of list

(DEFUN DISPLAY-REVERSE-LIST (LIST)
  (VIDEO-REVERSE)                                    ; Switch highlighting on
  (DISPLAY-LIST LIST)                                ; Display list
  (VIDEO-NORMAL))                                    ; Switch highlighting off

(DEFUN DISPLAY-LIST-XY (LIST X Y WINDOW)
  (GOTOXY X Y WINDOW)                                ; Position cursor
  (DISPLAY-LIST LIST))                               ; Print list on screen

(DEFUN DISPLAY-REVERSE-LIST-XY (LIST X Y WINDOW)
  (VIDEO-REVERSE)                                    ; Switch highlighting on
  (DISPLAY-LIST-XY LIST X Y WINDOW)                  ; Display list
  (VIDEO-NORMAL))                                    ; Switch highlighting off
```

Appendix 6k - Program Listings

```
(DEFUN DISPLAY-ASCII-LIST (LIST &OPTIONAL STREAM-NAME) ; Print list
  (COND ((NULL LIST))                                  ; End of list -> return NIL
        ((ATOM LIST) (PRINT-ASCII LIST
                                  STREAM-NAME))        ; Not list -> print element
        ((LISTP (CAR LIST))                            ; If still a list ->
           (DISPLAY-ASCII-LIST (CAR LIST)
                               STREAM-NAME)            ;   print first list
           (DISPLAY-ASCII-LIST (CDR LIST)
                               STREAM-NAME))           ;   print rest of lists
        (T (PRINT-ASCII (CAR LIST) STREAM-NAME)        ; Else print first element
           (DISPLAY-ASCII-LIST (CDR LIST)
                               STREAM-NAME))))         ;      print rest of list

(DEFUN PRINT-TEXT (TEXT-LIST &OPTIONAL STREAM-NAME) ; Print text list
  (COND ((NULL TEXT-LIST))                             ; End of list -> return NIL
        ((ATOM TEXT-LIST)                              ; One element =>
           (COND ((EQUAL TEXT-LIST *RETURNSYM*)        ; If return =>
                         (TERPRI STREAM-NAME))         ;    issue line-feed
                 (T (PRINT-ASCII TEXT-LIST
                                 STREAM-NAME))))       ; Else print element
        ((LISTP (CAR TEXT-LIST))                       ; If still a list ->
           (PRINT-TEXT (CAR TEXT-LIST)
                               STREAM-NAME)            ;   print first list
           (PRINT-TEXT (CDR TEXT-LIST)
                               STREAM-NAME))           ;   print rest of lists
        (T (COND ((EQUAL
                      (CAR TEXT-LIST) *RETURNSYM*)     ; If return =>
                      (TERPRI STREAM-NAME))            ;    issue line feed
                 (T (PRINT-ASCII (CAR TEXT-LIST)
                                 STREAM-NAME)))        ; Else print first element
           (PRINT-TEXT (CDR TEXT-LIST)
                               STREAM-NAME))))         ;      print rest of list

(DEFUN DISPLAY-ASCII-LIST-XY (LIST X Y WINDOW)
  (GOTOXY X Y WINDOW)                                  ; Position cursor
  (DISPLAY-ASCII-LIST LIST))                           ; Display ascii list

(DEFUN DISPLAY-REVERSE-ASCII-LIST (LIST)
  (VIDEO-REVERSE *TERMINAL-IO*)                        ; Switch highlighting on
  (DISPLAY-ASCII-LIST LIST)                            ; Display ascii list
  (VIDEO-NORMAL *TERMINAL-IO*))                        ; Switch highlighting off

(DEFUN DISPLAY-REVERSE-ASCII-LIST-XY (LIST X Y WINDOW)
  (GOTOXY X Y WINDOW)                                  ; Position cursor
  (VIDEO-REVERSE *TERMINAL-IO*)                        ; Switch highlighting on
  (DISPLAY-ASCII-LIST LIST)                            ; Display ascii list
  (VIDEO-NORMAL *TERMINAL-IO*))                        ; Switch highlighting off

(DEFUN ASK-CORRECT NIL
  (SCREEN-BOTTOM *MESS-QUERY-1*)                       ; Ask for confirmation
  (YES-NO-P))                                          ; Read yes or no

(DEFUN ASK-HALT NIL
  (SCREEN-BOTTOM *MESS-QUERY-2*)                       ; Ask for confirmation
  (YES-NO-P))                                          ; Read yes or no
```

```
DEFUN YES-NO-P NIL
 (LET ((ANSWER (SEND *QUERY-IO* :READ-CHAR)))      ; Read character
    (COND ((OR (EQUAL ANSWER #\N)                  ; If "N" or
               (EQUAL ANSWER #\n)) NIL)            ;   "n" -> return NIL
          ((OR (EQUAL ANSWER #\Y)                  ; Else if "Y" or
               (EQUAL ANSWER #\y)) T)              ;   "y" -> return TRUE
          (T (YES-NO-P)))))                        ; Else try again

DEFUN SCREEN-BOTTOM (MESSAGE)
 (VIDEO-REVERSE *MESSAGE-WINDOW*)                  ; Switch highlighting on
 (GOTOXY 0 0 *MESSAGE-WINDOW*)                     ; Position cursor
 (CLEAR-EOL *MESSAGE-WINDOW*)                      ; Set bottom
 (DISPLAY-XY MESSAGE 3 0 *MESSAGE-WINDOW*)         ; Display message in box
 (VIDEO-NORMAL *MESSAGE-WINDOW*))                  ; Switch highlighting off

DEFUN REMOVE-BOTTOM NIL
 (CLEAR-SCREEN *MESSAGE-WINDOW*))                  ; Clear message window

DEFUN RULE-BOTTOM (MESSAGE)
 (VIDEO-REVERSE *RULE-WINDOW*)                     ; Switch highlighting on
 (GOTOXY 0 0 *RULE-WINDOW*)                        ; Position cursor
 (CLEAR-EOL *RULE-WINDOW*)                         ; Set bottom
 (DISPLAY-XY MESSAGE 3 0 *RULE-WINDOW*)            ; Display message in box
 (VIDEO-NORMAL *RULE-WINDOW*))                     ; Switch highlighting off

DEFUN REMOVE-RULE-BOTTOM NIL
 (CLEAR-SCREEN *RULE-WINDOW*))                     ; Clear message window

DEFUN PRINT-ASCII (CODE &OPTIONAL STREAM-NAME)
 (PRINC (STRING CODE) STREAM-NAME))                ; Print ascii character

DEFUN BEEP NIL
 (%SYSINT #X21 #X0600 0 0 7)                       ; Make a beep
 NIL)                                              ; Return NIL
```

```
;;=============================================================================
;;FILE NAME:  RULEBASe
;;
;;  This group of functions prompt the user for details of the rule base
;;  which will describe the desired scanning technique.
;;=============================================================================

(DEFUN KEEP-CURRENT-RULE-BASE NIL
  (SCREEN-BOTTOM (STRING-APPEND *TECHNIQUE*
                                *MESS-QUERY-9*))    ; Ask if current base okay
  (LET ((ANSWER (YES-NO-P)))                        ; T or NIL
       (REMOVE-BOTTOM) ANSWER))                     ; Clean up => T or NIL

(DEFUN CREATE-NEW-BASE NIL
  (COND ((READ-IN-RULE-BASE)                        ; If file of rules read in
         (SETQ *RULE-BASE* NIL)                     ; Reset rule base to nil
         (DEFINE-BASIC-RULES)                       ; Set basic rules
         (INITIALISE-GROUPS)                        ; Initialise new technique
         (COND ((KEEP-CURRENT-RULE-BASE) T)         ;   Ask for confirmation=>T
               (T NIL)))                            ;   Else return NIL
        (T NIL)))                                   ; Else return NIL

(DEFUN READ-IN-RULE-BASE NIL
  (LET ((NAME (CHECK-DIR
               *MESSAGE-WINDOW* *MESS-PROMPT-4*)))  ; Read a name
       (COND ((NULL (PROBE-FILE
                     (PATHNAME NAME)))              ; If file does not exist
              (FILE-ERROR NAME *MESS-PROMPT-3*);          Report file error
              NIL)                                  ;       Return NIL
             (T (SCREEN-BOTTOM (STRING-APPEND NAME ; Else report loading
                                *MESS-INFO-1*))
              (LOAD NAME :VERBOSE NIL)              ; Else load the file
              (CLEAR-SCREEN *TERMINAL-IO*)          ;       Clear monitor
              T))))                                 ;       and return T

(DEFUN REPORT-NO-TECHNIQUE NIL
  (SCREEN-BOTTOM *MESS-PROMPT-5*)                   ; Display message
  (VIDEO-REVERSE *MESSAGE-WINDOW*)                  ; Switch highligting on
  (CLEAR-INPUT *MESSAGE-WINDOW*)                    ; Clear input buffer
  (READ-CHAR     *MESSAGE-WINDOW*)                  ; Read a character
  (VIDEO-NORMAL  *MESSAGE-WINDOW*)                  ; Switch highligting off
  (REMOVE-BOTTOM))                                  ; Clean up
```

```
;;=============================================================================
;;FILE NAME:  SCANNER
;;
;;  This group of functions define the scanning of a group of elements.
;;=============================================================================

;;=============================================================================
;;  SECTION I:  Scanning of one or more groups.
;;=============================================================================

(DEFUN SCAN (GROUP)
  (COND
    ((NULL GROUP) NIL)                                  ; No group -> return NIL
    (T (COND                                            ; Check if to scan group
         ((COND (*MULTIPLE-GROUPS*                      ; If multiples ->
                     (SCAN-GROUP-P GROUP))              ;      query scan
                (T))                                    ; Else default to true
           (SCAN-GROUP GROUP))                          ;  Go ahead with scan
         (T NIL)))))                                    ;  Else return NIL

(DEFUN SCAN-GROUP-P (GROUP)
  (VIDEO-REVERSE (GET-GROUP GROUP :WINDOW))             ; Switch highlighting on
  (DISPLAY-FRAME (GET-GROUP GROUP :STARTX)              ; Circle group with X coord
                 (GET-GROUP GROUP :ENDX)                ;  identify ending X coord
                 (GET-GROUP GROUP :STARTY)              ;  identify start Y coord
                 (GET-GROUP GROUP :ENDY)                ;  identify ending Y coord
                 (GET-GROUP GROUP :WINDOW))             ;  identify window
  (VIDEO-NORMAL  (GET-GROUP GROUP :WINDOW))             ; Switch highlighting off
  (LET ((CHOICE
         (KEYPRESSED (GET-GROUP GROUP :GROUP))))        ; Look for selection
     (DISPLAY-FRAME (GET-GROUP GROUP :STARTX)           ; Circle group with X coord
                    (GET-GROUP GROUP :ENDX)             ;  identify ending X coord
                    (GET-GROUP GROUP :STARTY)           ;  y starting Y coord
                    (GET-GROUP GROUP :ENDY)             ;  identify ending Y coord
                    (GET-GROUP GROUP :WINDOW))          ;  identify window
     CHOICE))                                           ; Return selection result

(DEFUN SCAN-GROUP (GROUP)
  (COND (*MULTIPLE-GROUPS* (SCAN-A-GROUP GROUP))        ; groups? => carry on
        (T (LET ((ITEM (SCAN-A-GROUP GROUP)))           ; Scan group
             (COND ((NULL ITEM) (SCAN-GROUP GROUP))     ; If no selection => rescan
                   (T ITEM))))))                        ; Else return item

(DEFUN SCAN-A-GROUP (GROUP)
  (COND ((EQUAL (GET-GROUP GROUP :ROW) 1)               ; Only one row ->
         (CAAR                                          ; Identify element returned
           (SCAN-COLUMN                                 ; by Choosing GROUP colum:
             (GET-GROUP GROUP :GROUP)                   ;         group
             (GET-GROUP GROUP :STARTX)                  ;         starting X coord
             (GET-GROUP GROUP :STARTY)                  ;         starting Y coord
             (GET-GROUP GROUP :WINDOW))))               ;         window
        (T (SCAN-ROW                                    ; Else scan the rows of the
             (SCAN-COLUMN                               ;   chosen column of GROUP:
               (GET-GROUP GROUP :GROUP)                 ;         group
               (GET-GROUP GROUP :STARTX)                ;         starting X coord
               (GET-GROUP GROUP :STARTY)                ;         starting Y coord
               (GET-GROUP GROUP :WINDOW)))))))          ;         window
```

```
Appendix 6m - Program Listings
(DEFUN SCAN-ROW (PARAMETERS)
   (LET ((ROW (CAR PARAMETERS))                          ; Assign list head to row
         (X (CADR PARAMETERS))                           ; Assign 2nd element to X
         (Y (CADDR PARAMETERS))                          ; Assign 3rd element to Y
         (WINDOW (CADR (CDDR PARAMETERS))))              ;   " 4th element to window
    (COND ((NULL ROW) NIL)                               ; No list -> return NIL
          ((EQUAL (LENGTH ROW) 0) (CAR ROW))             ; No elements -> choice
          (T                                             ; Else
  (LET ((CHOICE (SCAN-THE-ROW ROW X Y WINDOW)))          ; Scan the row
     (COND ((NULL CHOICE)                                ; No choice => reselect
       (COND ((ESCAPE-GROUP ROW X (- Y *GAPY*)           ;   ?reselection => escape
                            WINDOW) NIL)
             (T (SCAN-ROW PARAMETERS))))                 ;   Else rescan
       (T CHOICE)))))))                                  ; Else return choice

(DEFUN SCAN-THE-ROW (ROW X Y WINDOW)
  (COND ((EQUAL (LENGTH ROW) 0) NIL)                     ; Row empty -> return NIL
    (T (VIDEO-REVERSE WINDOW)                            ; Switch highlighting on
       (DISPLAY-XY (CAR ROW) X Y WINDOW)                 ; Display first element
       (VIDEO-NORMAL WINDOW)                             ; Switch highlighting off
       (LET ((SELECT (KEYPRESSED (CAR ROW))))            ; Look for switch selection
         (DISPLAY-XY (CAR ROW) X Y WINDOW)               ; redisplay row
         (COND ((EQUAL SELECT T)                         ; Switch activation
                   (CAR ROW))                            ;    -> return element
               (T (SCAN-THE-ROW                          ; Else call row scan with:
                           (CDR ROW)                     ;   rest of rows
                            X                            ;   column count
                           (+ Y *GAPY*)                  ;   increment row count
                            WINDOW)))))))                ;   window

(DEFUN SCAN-COLUMN (COLUMN X Y WINDOW)
  (COND ((NULL COLUMN) NIL)                              ; No list -> return NIL
        (T                                               ; Else
 (LET ((CHOICE (SCAN-THE-COLUMN COLUMN X Y WINDOW)))     ; Scan the column
    (COND ((NULL CHOICE)                                 ; No choice -> reselect
           (COND ((ESCAPE-GROUP COLUMN X (- Y *GAPY*)
                                WINDOW) NIL)             ; If selected => escape
                 (T (SCAN-THE-COLUMN COLUMN              ; Else rescan
                                X Y WINDOW))))
          (T CHOICE))))))                                ; Else return choice

(DEFUN SCAN-THE-COLUMN (COLUMN X Y WINDOW)
  (COND ((EQUAL (LENGTH COLUMN) 0) NIL)                  ; No columns -> return NIL
        (T (VIDEO-REVERSE WINDOW)                        ; Else highlighting on
           (DISPLAY-COLUMN (CAR COLUMN)                  ;   display first column
                           X Y WINDOW)
           (VIDEO-NORMAL WINDOW)                         ;   switch highlighting off
           (LET ((SELECT (KEYPRESSED (CAR COLUMN))))     ;     look for switch
             (DISPLAY-COLUMN (CAR COLUMN)                ;    redisplay column
                             X Y WINDOW)
             (COND ((EQUAL SELECT T)                     ; If switch activated ->
                    (LIST                                ;    Return list of:
                    (CAR COLUMN) X Y WINDOW))            ;   selected column & coord
                   (T (SCAN-THE-COLUMN (CDR COLUMN)      ; Else scan remaining cols.
                                       (+ X *GAPX*)      ; incrementing column count
                                        Y                ;    leaving Y as is
                                       WINDOW))))))))    ;    window
```

```
;;=============================================================================
;;  SECTION II:  Functions which display groups on the screen.
;;=============================================================================

(DEFUN DISPLAY-GROUP (NAME)
  (DISPLAY-THE-GROUP (GET-GROUP NAME :GROUP)        ; Identify group
                     (GET-GROUP NAME :STARTX)       ; Identify starting X coord
                     (GET-GROUP NAME :STARTY)       ; Identify starting Y coord
                     (GET-GROUP NAME :WINDOW)))     ; Identify window

(DEFUN DISPLAY-THE-GROUP (ELEMENTS STARTX STARTY WINDOW)
  (COND ((NULL ELEMENTS))                           ; End of element list
        (T (DISPLAY-COLUMN    (CAR ELEMENTS)        ; Call SCAN2 with first row
                              STARTX STARTY         ;   row count and new colum

                              WINDOW)               ;   in window
           (DISPLAY-THE-GROUP (CDR ELEMENTS)        ; Call again with rest
                              (+ STARTX *GAPX*)     ;   increment column count
                              STARTY WINDOW))))     ;   pass row count in windo


(DEFUN DISPLAY-COLUMN (ELEMENTS X Y WINDOW)
  (COND ((EQUAL (LENGTH ELEMENTS) 0) NIL)           ; End of row
   (T (DISPLAY-ELEMENT (CAR ELEMENTS) X Y           ; Else print first element
                                      WINDOW)       ;    in window
      (DISPLAY-COLUMN  (CDR ELEMENTS)               ; Display rest of elements
                       X (+ Y *GAPY*)               ; using same column, new ro

                       WINDOW))))                   ;   in window

(DEFUN DISPLAY-ELEMENT (ELEMENT X Y WINDOW)
  (DISPLAY-XY ELEMENT X Y WINDOW))                  ; Display element at X Y

(DEFUN DISPLAY-FRAME (STARTX ENDX STARTY ENDY WINDOW)
  (DO ((X (- STARTX 2) (+ X 1)))                    ; Draw top border
     ((EQUAL X (+ ENDX 4)))
    (DISPLAY-XY *BLANK* X (- STARTY 1) WINDOW))
  (DO ((Y STARTY (+ Y 1)))                          ; Set row count for body
     ((EQUAL Y (+ ENDY 1)))                         ; Set row limit
    (DO ((X (- STARTX 2) (+ ENDX 3))                ; Set column count for body
         (LOOP 1 (+ LOOP 1)))                       ; Set loop counter
      ((EQUAL LOOP 3))                              ; Set column limit
      (DISPLAY-XY *BLANK* X Y WINDOW)))             ;    draw body
  (DO ((X (- STARTX 2) (+ X 1)))                    ; Draw bottom border
     ((EQUAL X (+ ENDX 4)))
    (DISPLAY-XY *BLANK* X (+ ENDY 1) WINDOW)))

;;=============================================================================
;;  SECTION III:  Allowing for reselection of group/column/row.
;;=============================================================================

(DEFUN ESCAPE-GROUP (LIST X Y WINDOW)
  (KEYPRESSED LIST))                                ; Read selection
```

```
;;=============================================================================
;FILE NAME:  RULE MANIpulations
;;
;;  This group of functions define the procedures used to add, delete and
;;  modify rules in the rule base.
;;=============================================================================

(DEFMACRO REMOVE-RULE (POSITION RULE-BASE)
  `(SETQ ,RULE-BASE                                  ; Reset base to original
     (REMOVE-AT-POSITION ,POSITION ,RULE-BASE)))     ; w/o rule

(DEFMACRO ADD-RULE (RULE POSITION RULE-BASE)
  `(COND ((NULL ,RULE) NIL)                          ; No rule => NIL
         (T (SETQ ,RULE-BASE                         ; Else add new rule
           (ADD-LIST ,RULE ,POSITION ,RULE-BASE))))); with new rule

(DEFUN REPLACE-RULE (POSITION NEW-RULE)
   (COND ((NULL POSITION) NIL)                       ; If no rule => NIL
         ((NULL NEW-RULE) NIL)                       ; If no new rule => NIL
         (T (REMOVE-RULE POSITION *RULE-BASE*)       ; Else remove & add
            (ADD-RULE NEW-RULE POSITION *RULE-BASE*))))

(DEFUN VIEW-RULE (RULE)
   (COND ((NULL RULE) NIL)                           ; If no rule => NIL
         (T (REMOVE-RULE-BOTTOM)                     ; Clear window
            (RULE-BOTTOM *MESS-PROMPT-13*)           ; set prompt
            (GOTOXY 0 1 *RULE-WINDOW*)               ; Position cursor
            (DISPLAY-THE-RULE RULE)                  ; Print rule
            (READ-CHAR *RULE-WINDOW*) NIL)))         ; Wait

(DEFUN EXAMINE-LIST (E-LIST LEVEL)
  (LET ((IN-KEY (READ-CHAR)))                        ; Read from keyboard
    (COND ((EQUAL IN-KEY 24)                         ; If up, decrement level
           (EXAMINE-LIST E-LIST (GO-THRU-LIST E-LIST (DECF LEVEL))))
          ((EQUAL IN-KEY 25)                         ; If down, increment level
           (EXAMINE-LIST E-LIST (GO-THRU-LIST E-LIST (INCF LEVEL))))
          ((EQUAL IN-KEY *ESCSYM*) LEVEL)            ; If esc, return  level
          (T (EXAMINE-LIST E-LIST LEVEL)))))         ; Else go back
```

```
DEFUN GO-THRU-LIST (LIST LEVEL)
  (COND ((< LEVEL 0)                                  ; If top of list
         (DISPLAY-RULE *NULSYM* 1 NIL)                ;   Print #
         (INCF LEVEL))                                ;   Increment to 0
        ((= LEVEL (+ (LENGTH LIST) 1))                ; If end of list
         (DECF LEVEL)))                               ; Decrement level
  (COND ((= LEVEL 0)                                  ; If top of list
         (DISPLAY-RULE *NULSYM* 1 NIL))               ;   Print #
        (T (DO ((LEV 0 (INCF LEV))                    ; Else initialise counter
                (LST LIST (CDR LST)))                 ; Go through list
               ((EQUAL LEV (- LEVEL 1))               ; Higher level found =>
                (DISPLAY-RULE (CAR LST) 1 NIL)))))    ; display rule
  (DO ((LEV 0 (INCF LEV))                             ; Else initialise counter
       (LST LIST (CDR LST)))                          ; Go through list
      ((EQUAL LEV LEVEL)                              ; When level found =>
       (DISPLAY-RULE (CAR LST) 2 T)))                 ; display rule
  (COND ((= LEVEL (+ (LENGTH LIST) 1))                ; If end of list
         (DISPLAY-RULE *BLANK* 2 T)                   ; Print blank
         (DISPLAY-RULE *NULSYM* 3 NIL))               ; Print #
        ((= LEVEL (LENGTH LIST))                      ; If end of list
         (DISPLAY-RULE *NULSYM* 3 NIL))               ; Print #
        ((= LEVEL (- (LENGTH LIST) 1))                ; If end of list
         (DISPLAY-RULE *NULSYM* 3 NIL))               ; Print #
        (T (DO ((LEV 0 (INCF LEV))                    ; Else initialise counter
                (LST LIST (CDR LST)))                 ; Go through list
               ((EQUAL LEV (+ LEVEL 1))               ; Lower level found =>
                (DISPLAY-RULE (CAR LST) 3 NIL)))))    ; display rule,return level
  LEVEL)                                              ; Return level

DEFUN DISPLAY-RULE (RULE LINE REV)                    ; Print rule on screen
  (GOTOXY 0 LINE *RULE-WINDOW*)                       ; Position cursor
  (CLEAR-EOL  *RULE-WINDOW*)                          ; Clear line
  (GOTOXY 0 LINE *RULE-WINDOW*)                       ; Position cursor
  (SEND *RULE-WINDOW* :AUTO-NEWLINE NIL)              ; Auto-wrapping off
  (COND (REV (VIDEO-REVERSE *RULE-WINDOW*))           ; Switch highlight on
        (T ))
  (DISPLAY-THE-RULE RULE)                             ; Enter iteration
  (SEND *RULE-WINDOW* :AUTO-NEWLINE T)                ; Auto-wrapping on
  (VIDEO-NORMAL *RULE-WINDOW*))                       ; Switch highlight off

DEFUN DISPLAY-THE-RULE (RULE)
  (COND ((NULL RULE))                                 ; End of rule -> return NIL
        ((ATOM RULE)
               (PRINC RULE *RULE-WINDOW*)
               (PRINC *BLANK* *RULE-WINDOW*))         ; list -> print rule
        (T (PRINC (CAR RULE) *RULE-WINDOW*)
           (PRINC *BLANK* *RULE-WINDOW*)              ; first element
           (DISPLAY-THE-RULE (CDR RULE)))))           ;   print rest of rule

DEFUN LOOK-AT-RULE (RULES POSITION)
   (REMOVE-RULE-BOTTOM)
   (RULE-BOTTOM *MESS-PROMPT-11*)                     ; Explain operation
   (GO-THRU-LIST *RULE-BASE* POSITION)                ; Display current rule
   (EXAMINE-LIST RULES POSITION))                     ; Start examination
```

```
(DEFUN MODIFY-RULE (LEVEL FLAG)
  (LET ((POSITION (LOOK-AT-RULE *RULE-BASE* LEVEL)); Find level of interest
        (PROMPT (RULE-BOTTOM *MESS-PROMPT-10*))     ; Prompt
        (CLEAN1 (DISPLAY-RULE *BLANK* 1 NIL))
        (CLEAN2 (DISPLAY-RULE *BLANK* 3 NIL))       ; and clear
        (CODE (READ-CHAR *RULE-WINDOW*)))           ; Read from keyboard
    (DISPLAY-RULE *BLANK* 2 NIL)
    (DISPLAY-RULE (FIND-LIST-AT-LEVEL POSITION *RULE-BASE*) 3 T)
        (COND
  ((EQUAL POSITION (LENGTH *RULE-BASE*))  ;; If at end of rule, look for add.
                (COND ((OR (EQUAL CODE #\A) (EQUAL CODE #\a))
                           (MODIFY-RULE POSITION
                               (ADD-RULE (READ-A-RULE) POSITION *RULE-BASE*)))
                      (T (MODIFY-RULE POSITION FLAG))))
  ((OR (EQUAL CODE #\D) (EQUAL CODE #\d))           ; Look for deletion
     (MODIFY-RULE POSITION (REMOVE-RULE POSITION *RULE-BASE*)))
  ((OR (EQUAL CODE #\R) (EQUAL CODE #\r))           ; Look for replace
     (MODIFY-RULE POSITION (REPLACE-RULE POSITION (READ-A-RULE))))
  ((OR (EQUAL CODE #\A) (EQUAL CODE #\a))           ; Look for add.
     (MODIFY-RULE POSITION (ADD-RULE (READ-A-RULE) POSITION *RULE-BASE*)))
  ((OR (EQUAL CODE #\V) (EQUAL CODE #\v))           ; Look for view.
     (VIEW-RULE (FIND-LIST-AT-LEVEL POSITION *RULE-BASE*))
                (MODIFY-RULE POSITION FLAG))
  ((OR (EQUAL CODE #\E) (EQUAL CODE #\e))           ; Look for note.
     (VIEW-RULE (NOTE-ENVIRONMENT))
                (MODIFY-RULE POSITION FLAG))
  ((EQUAL CODE *ESCSYM*) FLAG)                      ; Look for escape.
  (T (MODIFY-RULE POSITION FLAG)))))                ; Else continue

(DEFUN READ-A-RULE NIL
  (RULE-BOTTOM *MESS-PROMPT-9*)                     ; Print prompt
  (READ-LINE-SYMBOL 0 1 *RULE-WINDOW*))             ; Read symbol
```

```
;;=============================================================================
;;FILE NAME:  STATiStics
;;  This group of functions prepare an output of statistics on the recent
;;  editing process.
;;=============================================================================

(DEFUN INIT-STATS NIL

                        ; Stats report messages
                        ; ======================

  (DEFCONSTANT *MESS-STATS-1*
     "                  COMMUNICATION RATE AND EFFICIENCY REPORT")
  (DEFCONSTANT *MESS-STATS-2*
     "~%                  ========================================")
  (DEFCONSTANT *MESS-STATS-3*
     "~%~%Anticipatory Algorithm:  ~S ~%Simulation File:       ~s")
  (DEFCONSTANT *MESS-STATS-4*
     "~%~%Processed Text to be found in file called:     ")
  (DEFCONSTANT *MESS-STATS-5*
     "~%Mirrored Text to be found in file called:      ")
  (DEFCONSTANT *MESS-STATS-6*
     "~%Number of elements chosen:      ")
  (DEFCONSTANT *MESS-STATS-7*
     "~%Number of deletions:              ")
  (DEFCONSTANT *MESS-STATS-8*
     "    Deletions per element:      ")
  (DEFCONSTANT *MESS-STATS-9*
     "~%                                 ------")
  (DEFCONSTANT *MESS-STATS-10*
     "~%      Total chosen:              ")
  (DEFCONSTANT *MESS-STATS-11*
     "~%~%Number of selections:            ")
  (DEFCONSTANT *MESS-STATS-12*
     "    Selections per element:     ")
  (DEFCONSTANT *MESS-STATS-13*
     "~%Number of scans not selected:   ")
  (DEFCONSTANT *MESS-STATS-14*
     "    No selections per element:  ")
  (DEFCONSTANT *MESS-STATS-15*
     "~%      Total scan:                ")
  (DEFCONSTANT *MESS-STATS-15A*
     "       Scan / No scan       : ")
  (DEFCONSTANT *MESS-STATS-16*
     "~%~%Observed Efficiency             ")
  (DEFCONSTANT *MESS-STATS-21*
     "~%Expected  Efficiency             ")
  (DEFCONSTANT *MESS-STATS-22*
     "     Difference:  ")
  (DEFCONSTANT *MESS-STATS-17*
     "  scans per element")
  (DEFCONSTANT *MESS-STATS-18*
     "~%Date:   ~s ")
  (DEFCONSTANT *MESS-STATS-19*
     " ~s   Duration:  ~s hours ~s minutes   Delay factor: ~s")
  (DEFCONSTANT *MESS-STATS-20*
     "~%Modified rule base to be found in file called: ")
```

```
                        ;; Messages for rule storage

  (DEFCONSTANT *MESS-STATS-30*
     ";; RULE BASE FROM ~s    -   ~s~%~%")
  (DEFCONSTANT *MESS-STATS-31*
     "(DEFUN INITIALISE-GROUPS NIL~%  (SETQ *TECHNIQUE* '~s)")
  (DEFCONSTANT *MESS-STATS-32*
     "  (SETQ *MODIFY-RULE* ~s)~%  (SETQ *MULTIPLE-GROUPS* ~s)")
  (DEFCONSTANT *MESS-STATS-33*
     "  (SETQ *MAXEDITROW* ~s)")
  (DEFCONSTANT *MESS-STATS-34*
    "~%  (SETQ *BASE-DELAY* ~s)")
  (DEFCONSTANT *MESS-STATS-35*
     "  (SETQ *MAX-GROUPS* ~s)")
  (DEFCONSTANT *MESS-STATS-36*
    "~%  (MAKE-GROUP-VARIABLE '~s~%      '(~s) ~s ~s)")
  (DEFCONSTANT *MESS-STATS-37*
    "~%  (SETQ *HIERARCHY* '(~s))")
  (DEFCONSTANT *MESS-STATS-38*
    "~%(REMEMBER *RULE-BASE* ~%   (~s~%      ~s) )~%~%") )

(DEFUN DISPLAY-STATISTICS (STREAM-NAME)

  (SETQ *TIME* (TIME-ELAPSED (GET-TIME) *TIME*))    ; Calculate time

  (FORMAT STREAM-NAME *MESS-STATS-1*)               ; Print heading
  (FORMAT STREAM-NAME *MESS-STATS-2*)               ;   underline


  (FORMAT STREAM-NAME *MESS-STATS-3* *TECHNIQUE*    ; display selection method,
          (COND ((SIMULATION-MODE-P) *SIMULATION-NAME*)
                (T *PROCESSING-MODE*)))

  (FORMAT STREAM-NAME *MESS-STATS-18* (CAR *DATE*)) ; Print day
     (PRINC (CADR *DATE*) STREAM-NAME)               ; Print month
  (FORMAT STREAM-NAME *MESS-STATS-19*
                                  (CADDR *DATE*)  ; Print year
                                    (CAR *TIME*)  ; Print hours
                                   (CADR *TIME*)  ; Print minutes
                                       *DELAY*)   ; and delay

  (FORMAT STREAM-NAME                               ; Print file for new text
     (STRING-APPEND *MESS-STATS-4* *RUN-NAME* ".NEW"))

  (FORMAT STREAM-NAME                               ; Print file for actual txt
     (STRING-APPEND *MESS-STATS-5* *RUN-NAME* ".MIR"))

  (FORMAT STREAM-NAME                               ; Print file for rule base
     (STRING-APPEND *MESS-STATS-20* *RUN-NAME* ".RUL"))

  (FORMAT STREAM-NAME "~%")                         ; Leave a blank line
  (FORMAT STREAM-NAME *MESS-STATS-6*)               ; Report total of elements
     (ALIGN-NUMBER *TOTAL-ELEMENTS* STREAM-NAME)

  (FORMAT STREAM-NAME *MESS-STATS-7*)               ; Report number deletions
     (ALIGN-NUMBER *KEYSTROKES-DEL* STREAM-NAME)
```

```
  (FORMAT STREAM-NAME *MESS-STATS-8*)                    ;   calculate average
     (COND ((OR (EQUAL *TOTAL-ELEMENTS* 0)
                (EQUAL *KEYSTROKES-DEL* 0))
                     (ALIGN-NUMBER 0 STREAM-NAME))
           (T (ALIGN-NUMBER (ROUND (/ *KEYSTROKES-DEL*
                                      *TOTAL-ELEMENTS*))
                     STREAM-NAME)))

  (FORMAT STREAM-NAME *MESS-STATS-9*)                    ; Underline for total
  (FORMAT STREAM-NAME *MESS-STATS-10*)                   ;   report total
     (ALIGN-NUMBER (+ *KEYSTROKES-DEL* *TOTAL-ELEMENTS*) STREAM-NAME)

  (FORMAT STREAM-NAME *MESS-STATS-11*)                   ; Report number of
     (ALIGN-NUMBER *KEYSTROKES-YES* STREAM-NAME)         ;   affirmative selections

  (FORMAT STREAM-NAME *MESS-STATS-12*)                   ;   calculate average
     (COND ((EQUAL *TOTAL-ELEMENTS* 0) (ALIGN-NUMBER 0 STREAM-NAME))
           (T (ALIGN-NUMBER (ROUND (/ *KEYSTROKES-YES*
                                      *TOTAL-ELEMENTS*))
                     STREAM-NAME)))

  (FORMAT STREAM-NAME *MESS-STATS-13*)                   ; Report number of
     (ALIGN-NUMBER *KEYSTROKES-NO* STREAM-NAME)          ;   negative scans

  (FORMAT STREAM-NAME *MESS-STATS-14*)                   ;   calculate average
     (COND ((EQUAL *TOTAL-ELEMENTS* 0) (ALIGN-NUMBER 0 STREAM-NAME))
           (T (ALIGN-NUMBER (ROUND (/ *KEYSTROKES-NO*
                                      *TOTAL-ELEMENTS*))
                     STREAM-NAME)))

  (FORMAT STREAM-NAME *MESS-STATS-9*)                    ; Underline for total
  (FORMAT STREAM-NAME *MESS-STATS-15*)                   ; Report total scans
  (ALIGN-NUMBER (TRUNCATE (* (+ (/ *KEYSTROKES-NO* 100)
                                (/ *KEYSTROKES-YES* 100))
                             100))
                STREAM-NAME)

  (FORMAT STREAM-NAME *MESS-STATS-15a*)                   ; Report scan ratio
  (ALIGN-NUMBER(ROUND(/ *KEYSTROKES-YES* *KEYSTROKES-NO*))STREAM-NAME)

  (SETQ *EFF* (COND ((EQUAL *TOTAL-ELEMENTS* 0) 0)
                    (T (/ (+ (/ *KEYSTROKES-YES* 100)
                             (/ *KEYSTROKES-NO*  100))
                          (/ *TOTAL-ELEMENTS* 100)))))

  (FORMAT STREAM-NAME *MESS-STATS-16*)                    ; Report efficiency
       (ALIGN-NUMBER (ROUND *EFF*) STREAM-NAME)
  (FORMAT STREAM-NAME *MESS-STATS-17*)

  (FORMAT STREAM-NAME *MESS-STATS-21*)
       (ALIGN-NUMBER *EFFICIENCY* STREAM-NAME)
  (FORMAT STREAM-NAME *MESS-STATS-22*)
       (ALIGN-NUMBER (- (ROUND *EFF*) *EFFICIENCY*) STREAM-NAME)


  (TERPRI STREAM-NAME)                                    ; End with new line
  )
```

```
(DEFUN REPORT-STATISTICS NIL

  (SETQ *TIME* (TIME-ELAPSED (GET-TIME) *TIME*))    ; Compute elapsed time

  (INIT-STATS)

  (STORE-RULES)

  (SETQ *KEYSTROKES-NO*   (- *KEYSTROKES-NO*
                             *CURRENT-NO*))         ; Else forget extra scans
  (SETQ *KEYSTROKES-YES* (- *KEYSTROKES-YES*
                            *CURRENT-YES*))

  (DECF *TOTAL-ELEMENTS*)

  (CLEAR-SCREEN *TERMINAL-IO*)                      ; Clear the screen
  (DISPLAY-STATISTICS *TERMINAL-IO*)                ; Display on screen

  (DISPLAY-STATISTICS *STATS-FILE*)                 ; Store statistics

  (VIDEO-REVERSE *MESSAGE-WINDOW*)
  (GOTOXY 0 3 *MESSAGE-WINDOW*)
  (CLEAR-EOL  *MESSAGE-WINDOW*)
  (DISPLAY-XY *MESS-QUERY-6* 0 3 *MESSAGE-WINDOW*) ; Offer printing
  (VIDEO-NORMAL *MESSAGE-WINDOW*)
      (COND ((YES-NO-P)                             ; Accept -> print stats
                (SCREEN-BOTTOM *MESS-QUERY-7*)      ; Check printer
                (READ-CHAR)                         ; Wait for keypress
                (OPEN-PRINTER *TEXT-FILE*)          ; Set up printer
                (DISPLAY-STATISTICS *TEXT-FILE*)    ; Write to print stream
                (CLOSE *TEXT-FILE*))                ; Print
            (T  ))                                  ; Else do nothing

  (SETQ *QUIT* NIL)                                 ; Reinitialise *QUIT*

  NIL)                                              ; End of function
```

```
;;=============================================================================
;;FILE NAME:  READDDIGRAM
;;
;;  This file specifies the functions used to read in and manipulate digram
;;  lists.
;;=============================================================================

;  Returns x number of items in a digram list
;-----------------------------------------------------------------------------

(DEFUN FIRST-BIT-DIGRAM (MEASURE)
  (COND ((NULL *DIGRAM-LIST*)                          ; If no more then
         (MATCH-A-DIGRAM (FIRST *NEWTEXT*)             ;   get new list
                         (CAR (LAST  *NEWTEXT*))))
        (T ))
  (DO ((LOOP 0 (INCF LOOP))                            ; Increment loop each time
       (NEWLIST NIL))                                  ; Initialise variable
      ((OR (NULL *DIGRAM-LIST*)                        ; End when list if empty
           (EQUAL LOOP MEASURE)) NEWLIST)              ;   or enough loops => list
    (COND ((EQUAL (CAR *DIGRAM-LIST*) *BLANK*)         ; If a blank => resplace
           (UPDATE-LIST NEWLIST *INSERTSYM*))          ;   with the symbol
          (T (UPDATE-LIST NEWLIST
              (CAR *DIGRAM-LIST*))))                   ; Else add item to new list
    (FORGET-FIRST *DIGRAM-LIST*)))                     ; Remove from old list

;-----------------------------------------------------------------------------
;  To read in digrams
;-----------------------------------------------------------------------------
(DEFUN SKIP-MANY-LINES (NUMBER FILE)
   (DO ((COUNT 1 (INCF COUNT))                         ; Read past next lines
        (NEWCHAR (READ-FILE-CHAR FILE)
                 (READ-FILE-CHAR FILE)))               ; Read next char
       ((OR (EQUAL COUNT NUMBER)                       ; stop after NUMBER of line
            (NULL  NEWCHAR))                           ; or end of file
         (COND ((EQUAL COUNT NUMBER) (READ-LINE FILE))
               (T )))
     (READ-LINE FILE)))                                ; read next line

(DEFUN SKIP-TILL-FIRST (FIRSTCH FILE)
   (READ-LINE FILE)                                    ; Flush end of line
   (DO ((NEWCHAR (READ-FILE-CHAR FILE)                 ; Read first characters
                 (READ-FILE-CHAR FILE)))
       ((COND ((EQUAL (STRING NEWCHAR) FIRSTCH)        ; stop if found
               (UNREAD-FILE-CHAR FILE NEWCHAR) T)      ;  and replace in file
              ((NULL NEWCHAR) T)                       ; or end of file
              (T NIL)) NEWCHAR)
     (READ-LINE FILE)))                                ; read next line

(DEFUN READ-LIST (DLIST)
  (LET ((ACHAR (READ-FILE-CHAR *DIGRAM-FILE*)))        ; Read the next character
    (COND ((OR (NULL ACHAR)                            ; ? End of file => list
               (EQUAL ACHAR 10)) DLIST)                ; ? End of line => list
          ((NULL DLIST)                                ; ? No digram yet
               (READ-LIST (LIST (STRING ACHAR))))      ;       => start with char
          (T (READ-LIST (APPEND DLIST                  ; Else call function with
                                (LIST (STRING ACHAR)) ;       new list
                        ))))))
```

```
Appendix 6p - Program Listings
(DEFUN READ-A-LIST NIL
  (DO ((NEWLIST (READ-LIST NIL) (READ-LIST NIL))    ; Read list every round
       (ALIST NIL))                                 ; Initialise new list
      ((OR (NULL NEWLIST)                           ; End if end of file
           (EQUAL NEWLIST 'T)) ALIST)               ;   or end of list => list
    (SETQ ALIST (APPEND ALIST (LIST NEWLIST)))))    ; append new list on end

(DEFUN READ-A-DIGRAM NIL
  (DO* ((LET1 (READ-FILE-CHAR *DIGRAM-FILE*)
              (READ-FILE-CHAR *DIGRAM-FILE*))       ; Read a char every round
        (NEWDIGRAM (READ-A-LIST) (READ-A-LIST))     ; Read a digram every round
        (DIGRAMS NIL))                              ; Initialise new list
       ((NULL LET1) DIGRAMS)                        ; End if end file => list
    (SETQ DIGRAM1                                   ; Append first character to
          (LIST (LIST (STRING LET1)) NEWDIGRAM))    ;    make first part
    (SETQ DIGRAMS
          (APPEND DIGRAMS (LIST DIGRAM1)))))        ; Append new part to old

(DEFUN MATCH-DIGRAM (FIRSTLET SECONDLET)
  (LET ((NEWCHAR (READ-FILE-CHAR *DIGRAM-FILE*)))   ; Read first char on line
    (COND ((NULL NEWCHAR) NIL)                      ; End of file => return
          ((EQUAL (STRING NEWCHAR) FIRSTLET)        ; Char = first letter =>
             (COND ((EQUAL (STRING (READ-FILE-CHAR ;   read next character
                                   *DIGRAM-FILE*))
                           SECONDLET)               ;   Char = second letter =>
                    (READ-FILE-LINE                 ;         return the line
                                   *DIGRAM-FILE*))
                   (T (READ-LINE *DIGRAM-FILE*)     ;   Else exclude line
                      (MATCH-DIGRAM FIRSTLET        ;        and try next line
                                    SECONDLET))))
          (T
  (COND ((NULL (SKIP-TILL-FIRST FIRSTLET *DIGRAM-FILE*)) ; Read till found
               NIL)                                 ; If end of file => nil
        (T (MATCH-DIGRAM FIRSTLET SECONDLET)))))))  ; Else try match again

(DEFUN OPEN-DIGRAM-FILE NIL
  (SETF *DIGRAM-FILE*                               ; Give stream a name
    (OPEN (PATHNAME *DIGRAM-NAME*)                  ; open a file
          :DIRECTION :INPUT)))                      ; for input

(DEFUN MATCH-A-DIGRAM (FIRSTLET SECONDLET)
  (COND ((NULL *DIGRAM-FILE*))                      ; If file open
        (T (CLOSE *DIGRAM-FILE*)))                  ; Make sure file is closed
  (OPEN-DIGRAM-FILE)                                ; Reopen the digram file
  (SETQ *DIGRAM-LIST*                               ; Look for the matching
        (MATCH-DIGRAM FIRSTLET SECONDLET)))         ;  digram

;-------------------------------------------------------------------------
;  Read in name of digram file, i.e. alphabet listings
;-------------------------------------------------------------------------

(DEFUN READ-IN-DIGRAM-NAME NIL
  (SCREEN-BOTTOM *MESS-PROMPT-8*)                   ; Display message
  (VIDEO-REVERSE *MESSAGE-WINDOW*)                  ; Switch highligting on
  (SETQ *DIGRAM-NAME* (READ *MESSAGE-WINDOW*))      ; Read a name
  (VIDEO-NORMAL *MESSAGE-WINDOW*)                   ; Switch highligting off
  (REMOVE-BOTTOM))                                  ; Clean up
```

```
;;=============================================================================
;;FILE NAME:  SIMULATion
;;  This group of functions determine whether the mode of processing text is
;;  is to be "free" entry or in a simulation mode.
;;=============================================================================
(DEFUN UPDATE-SIMULATION-FILE (CHOICE)
  (SETQ TEMP (READ-FILE-CHAR *SIMULATION-FILE*))     ; Retrieve next char
  (COND ((EQUAL TEMP (ASCII CHOICE) ))               ; If match do nothing
        (T  (UNREAD-FILE-CHAR *SIMULATION-FILE*
                              TEMP))))               ;  Else replace last read

(DEFUN SIFT-SIMULATION-RULES (ITEM RULES)
  (COND ((NULL RULES) (LIST 'NO-MATCH))              ; No rules => NIL
        ((EVAL (CAAR RULES)) (EVAL (CADAR RULES)))   ; If match => evaluation
        (T (SIFT-SIMULATION-RULES ITEM (CDR RULES)))))

(DEFUN TRANSPOSE-ITEM (ITEM)
  (SIFT-SIMULATION-RULES ITEM *SIMULATION-RULES*))   ; Call lower function

(DEFUN PROCESS-NEW-CHAR NIL
  (COND ((NULL (CAR *SIMULATION-LIST*)) )            ; If end - stop
        (*QUIT* )                                    ; If quit set -> stop
        ((INGROUP-P (STRING (CAR *SIMULATION-LIST*))
            (GET-GROUP (GET-ENVIRONMENT :GROUP)
                  :GROUP)) *SIMULATION-LIST*)        ; If item found =>
        ((EQUAL (CAR *SIMULATION-LIST*) 'NO-MATCH)   ; If no match =>
                        *SIMULATION-LIST*)
        (T (SETQ *SIMULATION-LIST*                   ; Else
              (TRANSPOSE-ITEM (STRING
                    (CAR *SIMULATION-LIST*))))       ; Change item
           (PROCESS-NEW-CHAR))))                     ; Check again

(DEFUN READ-NEXT-CHAR NIL
  (SETQ *SIMULATION-LIST* (LIST
       (SEND *SIMULATION-FILE* :READ-CHAR-NO-HANG))); Retrieve next character
  (SEND *SIMULATION-FILE* :UNREAD-CHAR
                    (CAR *SIMULATION-LIST*)))      ; Replace the character

(DEFUN SIMULATE-ONE-UNIT NIL
  (COND ((NULL *SIMULATION-LIST*) (READ-NEXT-CHAR)) ; Read next character
        (T NIL))                                    ; Else characters remain
  (COND ((NULL (CAR *SIMULATION-LIST*))
                         (SETQ *QUIT* T))           ; If end => stop
        (T (PROCESS-NEW-CHAR)
           (WRITE-ONE-UNIT)
           (SETQ *SIMULATION-LIST*
                    (CDR *SIMULATION-LIST*)))))     ; Reduce list

(DEFUN INITIATE-SIMULATION NIL
  (COND ((NULL (SETQ *SIMULATION-NAME*
                (OPEN-INPUT-FILE *SIMULATION-FILE*
                                 'DIRECT            ; Open the text file
                                 *MESSAGE-WINDOW*))));   If file error return
        (T                                          ;   Else continue
            (SETQ *DELAY* 0)                        ; Initialise counters
            (WRITE)                                 ; Process simulation text
            (CLOSE *SIMULATION-FILE*))))            ; Close the file
```

Appendix 6r - Program Listings

```
;;=============================================================================
;;FILE NAME:  CHECKDIGRAM
;;
;;  This file specifies the functions used to ckeck the transcription of the
;;  digram file.
;;=============================================================================

(DEFUN CHECK-DIGRAM-FILE NIL
  (SETQ *DIGRAM-NAME* "ALPH.D18")
  (OPEN-DIGRAM-FILE)
  (SETQ ALPHABET '(" " "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
                   "N" "O" "P" "Q" "R" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"))
  (SETQ ERRORS NIL)
  (DO ((DLIST (READ-LINE *DIGRAM-FILE*) (READ-LINE *DIGRAM-FILE*)))
      ((NULL DLIST) ERRORS)
   (PRINC ".")
   (SETQ ERRORS (APPEND ERRORS (CHECK-DIGRAM DLIST ALPHABET)))))

(DEFUN CHECK-DIGRAM (LINE TEST-LINE)
  (COND ((NULL TEST-LINE) NIL)
        ((> (CHECK-ALPHABET LINE (CAR TEST-LINE)) 1)
              (LIST LINE))
        (T (CHECK-DIGRAM LINE (CDR TEST-LINE)))))

(DEFUN CHECK-ALPHABET (LINE ALPH )
  (DO ((COUNT 0)
       (FOUND (STRING-SEARCH ALPH LINE 2)
              (STRING-SEARCH ALPH LINE (+ FOUND 1))))
      ((NULL FOUND) COUNT)
   (COND ((NULL FOUND))
         (T (SETF COUNT (INCF COUNT))))))


(CHECK-DIGRAM-FILE)
(PPRINT ERRORS)
```

Appendix 3s - Program Listings

```
;;=============================================================================
;;FILE NAME:  DEVALUE
;;
;;  These two functions make all variables and functions used valueless so
;;  that the memory does not overflow.  The variable are made valueless after
;;  each run, wheras the functions are only devalued at the end of a session.
;;=============================================================================

(DEFUN DEVALUE-GLOBAL NIL

  ;; Make all global constants, parameters and variables valueless.

  (MAPCAR 'MAKUNBOUND

     '(
         ;;  CONSTANTS
         ;;  =========

  *BASE-DELAY*          *WAIT-DELAY*      *RULE-DELAY*      *BASEX*
  *BASEY*               *MAXEDITPAGE*
  *MAXEDITPARA*         *WORD-LENGTH*
  *MAXRULEROW*          *MAXRULECOL*      *MAXMESSAGEROW*   *MAXMESSAGECOL*
  *MAXSCRNCOL*          *SCREENLENGTH*    *PUNCTUATION*     *SENTENCE-END*

  *PERIOD*  *BLANK*     *NULSYM*     *CURSOR*      *QUITSYM*   *INSERTSYM*
  *CARET*   *ASTERISK*  *PUNCTSYM*   *REPEATSYM*   *PRINTSYM*  *DELETESYM*
  *ESCSYM*  *SPEEDUP*   *SPEEDDOWN*  *RETURNSYM*   *EOL*       *EOF*
  *YES*     *NO*        *ESC*        *ENDDIG*      *BEGDIG*    *LOWU*
  *UPPU*    *QSET*


  ;;  Stats report messages

  *MESS-STATS-1*   *MESS-STATS-2*   *MESS-STATS-3*   *MESS-STATS-4*
  *MESS-STATS-5*   *MESS-STATS-6*   *MESS-STATS-7*   *MESS-STATS-8*
  *MESS-STATS-9*   *MESS-STATS-10*  *MESS-STATS-11*  *MESS-STATS-12*
  *MESS-STATS-13*  *MESS-STATS-14*  *MESS-STATS-15*  *MESS-STATS-16*
  *MESS-STATS-17*  *MESS-STATS-18*  *MESS-STATS-19*  *MESS-STATS-20*
  *MESS-STATS-21*

  ;; Printing messages

  *MESS-PRINT-1*   *MESS-PRINT-2*   *MESS-PRINT-3*   *MESS-PRINT-4*

  ;; Query messages

  *MESS-QUERY-1*   *MESS-QUERY-2*   *MESS-QUERY-3*   *MESS-QUERY-4*
  *MESS-QUERY-5*   *MESS-QUERY-6*   *MESS-QUERY-7*   *MESS-QUERY-8*
  *MESS-QUERY-9*   *MESS-QUERY-10*

  ;; Prompt messages

  *MESS-PROMPT-1*   *MESS-PROMPT-2*   *MESS-PROMPT-3*   *MESS-PROMPT-4*
  *MESS-PROMPT-5*   *MESS-PROMPT-6*   *MESS-PROMPT-7*   *MESS-PROMPT-8*
  *MESS-PROMPT-9*   *MESS-PROMPT-10*  *MESS-PROMPT-11*  *MESS-PROMPT-12*
  *MESS-PROMPT-13*
```

```
  ;; Information messages

  *MESS-INFO-1*     *MESS-INFO-2*

  ;; Heading messages

  *MESS-HEADING-1*  *MESS-HEADING-2*  *MESS-HEADING-3*  *MESS-HEADING-4*
  *MESS-HEADING-5*  *MESS-HEADING-6*

       ;; PARAMETERS
       ;; ==========

  *GAPX* *GAPY* *ENV* *DATE* *TIME1* *TIME2*

       ;; VARIABLES
       ;; ---------

  *XY-COORDS*        *NEWTEXT*              *KEYSTROKES-YES*    *CURRENT-YES*
  *KEYSTROKES-NO*    *CURRENT-NO*           *KEYSTROKES-DEL*    *TOTAL-ELEMENTS*
  *DELAY*            *DEL-AV*               *OLD-DEL-AV*        *TEXT-FILE*
  *NEW-TEXT-FILE*    *MIRRORED-TEXT-FILE*   *STATS-FILE*        *RULE-FILE*
  *DIGRAM-NAME*      *DIGRAM-FILE*          *SIMULATION-FILE*   *SIMULATION-LIST*
  *RULE-BASE*        *SIMULATION-RULES*     *RESTRICTIONS*      *HISTORY*
  *QUIT*             *RULE-MODIFY*          *RUN-NAME*          *TECHNIQUE*
  *ENV-STORE*        *GROUPS-NOT-WANTED*    *HIERARCHY*         *HIERARCHY-LIST*
  *DIGRAM-LIST*      *MAX-GROUPS*           *PROCESSING-MODE*   *EFFICIENCY*
  *MAXEDITROW*       *MAXEDITCOL*           *LASTWORD*          *NEWWORD*

     ))

   (SEND *TERMINAL-IO* :CLEAR-INPUT)

   NIL)

(DEFUN DEVALUE-FUNCTIONS NIL

  ;  All functions are made valueless

  (MAPCAR 'FMAKUNBOUND '(

;;=============================================================================
;;FILE NAME:  EDIT
;;=============================================================================

   DELETE-CHARACTER CASCADE-THRU-RULES CHECK-CONTINUATION EXAMINE-STATUS
   NEW-LINE PRINT-CHOICE PRINT-FN PROCESS-ELEMENT PROCESS-RULE REPEAT-SYMBOL
   SAVE-FN SIFT-RULES TESTRULE UPDATE-ALL WRITE WRITE-ONE-LINE WRITE-ONE-PAGE
   WRITE-ONE-PARA WRITE-ONE-PAGE

;;=============================================================================
;;FILE NAME:  INIT
;;=============================================================================

   DEFINE-BASIC-RULES DEFINE-CONSTANTS DEFINE-ENVIRONMENT DEFINE-MESSAGES
   DEFINE-PARAMETERS DEFINE-VARIABLES DISPLAY-END DISPLAY-TITLE
   INIT-ENVIRONMENT INITIALISE-GLOBAL
```

```
;;=============================================================================
;;FILE NAME:  MAIN
;;=============================================================================

   ASK-RESTART EXECUTE-SCANNER DETERMINE-MODE DETERMINE-RULE-BASE SCANNER

;;=============================================================================
;;FILE NAME:  READ
;;=============================================================================

   CHECK-KEYPRESS CHECK-SPEED CLEAR-INPUT KEYPRESSED READ-A-CHAR READ-A-STRING
   READ-A-SYMBOL READ-FILE-CHAR READ-FILE-LINE READ-INTEGER READ-LINE-SYMBOL
   READ-UNTIL-MARK SIMULATE UNREAD-FILE-CHAR

;;=============================================================================
;;FILE NAME:  UTILities
;;=============================================================================

   ADD-LIST ALIGN-NUMBER ASCII CHOP-OFF-LAST CLEAR-LIST CLS COMPUTE-LEVEL
   CONVERT-LIST-TO-STRING FIND-LIST-AT-LEVEL FORGET-FIRST FORGET-LAST
   GROUP-NAME IN-GROUP-P LIST-TO-STRING MEMBER= LOAD-FILE OPPOSITE-CASE
   REMEMBER REMOVE-AT-POSITION REMOVE-IF-EQUAL ROUND SECOND-BUT-LAST
   TACK-ON-LIST UPDATE-LIST

;;=============================================================================
;;FILE NAME:  STATiStics
;;=============================================================================

   DISPLAY-STATISTICS REPORT-STATISTICS

;;=============================================================================
;;FILE NAME:  FILEIO
;;=============================================================================

   FILE-ERROR OPEN-INPUT-FILE OPEN-OUTPUT-FILE OPEN-PRINTER READ-TEXT-NAME
   SCAN-TEXT-NAME

;=============================================================================
;FILE NAME:  ACCESS
;=============================================================================

   SET-UP-GLOBAL-WINDOW

   CHECK-FORMAT GET-GROUP FIND-MAXROW MAKE-GROUP-VARIABLE MAXCOL MAXROW
   MODIFY-AN-ELEMENT REFORMAT-TO-ROWS UPDATE-GROUP

   FIND-POSSIBLE-GROUP GROUP-ATTEMPTED-P REMEMBER-GROUP

   GET-ENVIRONMENT GET-LAST-ENVIRONMENT FIND-ATTRIBUTE NOTE-ENVIRONMENT
   RECALL-ENVIRONMENT RECORD-ENVIRONMENT REINSTATE-ENVIRONMENT
   RESET-CONTINUE UPDATE-ENVIRONMENT

;=============================================================================
;FILE NAME:  CURSOR
;=============================================================================

   POSITION-WINDOW-CURSOR QUERY-WINDOW-POSITION QUERY-WINDOW-X-POSITION
                                                QUERY-WINDOW-Y-POSITION
```

```
   CURRENT-X CURRENT-Y RECORD-XY-POSITION REMEMBER-XY-POSITION
   REVERSE-XY-POSITION W UNDO-XY-POSITION

   ADVANCE-LINE ADVANCE-XY-POSITION DELETE-LAST-CHAR MOVE-ON POSITION-CURSOR
   REMOVE-CURSOR

;;============================================================================
;;FILE NAME:  RUNOUTput
;;============================================================================

   ACCEPT-EXISTING-NAMES CHECK-RUN-FILE CLOSE-ALL-FILES OPEN-ALL-RUN-FILES
   OPEN-RUN-FILES STORE-RULES

;;============================================================================
;;FILE NAME:  SCREEN
;;============================================================================

   ASK-CORRECT ASK-HALT BEEP CLEAR-EOL CLEAR-EOS CLEAR-SCREEN
   DISPLAY-ASCII-LIST-XY DISPLAY-REVERSE-ASCII-LIST
   DISPLAY-REVERSE-ASCII-LIST-XY DISPLAY-XY DISPLAY-REVERSE-XY
   DISPLAY-LIST DISPLAY-REVERSE-LIST DISPLAY-LIST-XY DISPLAY-REVERSE-LIST-XY
   DISPLAY-ASCII-LIST GOTOXY REMOVE-BOTTOM RULE-BOTTOM REMOVE-RULE-BOTTOM
   PRINT-ASCII PRINT-TEXT SCREEN-BOTTOM VIDEO-REVERSE VIDEO-NORMAL YES-NO-P

;;============================================================================
;;FILE NAME:  READDDIGRAM
;;============================================================================

   FIRST-BIT-DIGRAM MATCH-DIGRAM MATCH-A-DIGRAM OPEN-DIGRAM-FILE
   READ-LIST READ-A-LIST READ-A-DIGRAM READ-IN-DIGRAM-NAME SKIP-MANY-LINES

;;============================================================================
;;FILE NAME:  RULEBASe
;;============================================================================

   CREATE-NEW-BASE KEEP-CURRENT-RULE-BASE READ-IN-RULE-BASE
   REPORT-NO-TECHNIQUE

;;============================================================================
;;FILE NAME:  SCANNER
;;============================================================================

   SCAN SCAN-GROUP-P SCAN-GROUP SCAN-A-GROUP SCAN-ROW SCAN-THE-ROW
        SCAN-COLUMN SCAN-THE-COLUMN

   DISPLAY-COLUMN DISPLAY-ELEMENT DISPLAY-FRAME DISPLAY-GROUP
   DISPLAY-THE-GROUP

   ESCAPE-GROUP

;;============================================================================
;;FILE NAME:  SIMULATion
;;============================================================================

   INITIATE-SIMULATION PROCESS-NEW-CHAR READ-NEXT-CHAR
   SIFT-SIMULATION-RULES SIMULATE-ONE-UNIT SIMULATION-MODE-P TRANSPOSE-ITEM
   UPDATE-SIMULATION-FILE
```

Appendix 3s - Program Listings

```
;;=============================================================================
;;FILE NAME:  RULE MANIpulations
;;=============================================================================

   ADD-RULE DISPLAY-RULE DISPLAY-THE-RULE EXAMINE-LIST GO-THRU-LIST
   LOOK-AT-RULE MODIFY-RULE READ-A-RULE REMOVE-RULE REPLACE-RULE VIEW-RULE

;;=============================================================================
;;FILE NAME:  CHECKDIGRAM
;;=============================================================================

   CHECK-ALPHABET CHECK-DIGRAM CHECK-DIGRAM-FILE

;;=============================================================================
;;FILE NAME:  rule files - (these vary)
;;=============================================================================

   CREATE-DIGRAM-GROUP INITIATE-RULES INITIALISE-GROUPS

 ))
```