NASA CONTRACTOR REPORT

NASA CR-1634

DATA DISPLAY AND ANALYSIS

by J. Meads, D. DeFanti, and P. Sweet

Prepared by
WOLF RESEARCH & DEVELOPMENT CORPORATION
Bedford, Mass. 01730
for Goddard Space Flight Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JULY 1970

1. Report No.
NASA CR-1634

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
DATA DISPLAY AND ANALYSIS

5. Report Date
July 1970

6. Performing Organization Code

7. Author(s)
J. Meads, D. DeFanti, P. Sweet

8. Performing Organization Report No.
None

9. Performing Organization Name and Address
Wolf Res~~earch~~ & Dev~~elopment~~ Corp~~oration~~
P. O. Box 243, Crosby Drive
Bedford, Massachusetts 01730

10. Work Unit No.

11. Contract or Grant No.
NAS5-9756-151

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, D.C. 20546

13. Type of Report and Period Covered
Contractor Report

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This research is addressed to the problems of developing more efficient means by which the physical scientist might intimately control the complex data processing operations involved in the analysis and meaningful interpretation of large volumes of sensor data from scientific spacecraft. One such means is handwritten input of mathematical impressions, a most natural method for making changes to data analysis procedures, and to mathematical models, while working on-line with a computer. In the work reported here, adaptive recognizers of handprinted alphanumeric characters were studied, and one of these was implemented for the purpose of detailed investigation. This character recognizer featured the facility for defining special symbols by graphical construction. The investigation explored the general problem of communication between man and computer and identified the practical problems of the character recognizer.

1. Data Display

17. Key Words (Selected by Author(s))
Data display, computer graphics, display programming, character recognizers, man-machine interaction

18. Distribution Statement
Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
(U)	(U)	49	$3.00

*For sale by the Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia 22151.

ACKNOWLEDGEMENTS

This work was performed under the technical direction of Mr. D. E. Jamison GSFC, Code 565.

Implementation of the Bernstein character recognizer on the CDC 3200/IDIIOM system with GRAFACON Tablet was made possible through the courtesy of personnel of the Processor Development Branch (Code 563) GFSC.

Principal individual contributors to this effort included Mr. David DeFanti, Mr. Jon Meads and Mr. Phillip Sweet.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure | Page

1.0 INTRODUCTION

As a continuation of research directed toward techniques in interactive data analysis and display, investigations at Wolf Research and Development Corporation (WOLF) initiated design of a pilot study character recognition system in December 1968. The system was to be implemented on a CDC 3200 - IDIIOM display equipment configuration at the Goddard Space Flight Center. The work has been completed and is described in this report.

The section "MAN-MACHINE INTERACTION" provides a brief general discussion of the necessity for ease of communication between the computer and it's user, and makes note of the use of displays as the most efficacious means of achieving this state.

The section "SEMANTICS AND SYNTAX" presents some of the concepts necessary for effective communication. Formal definitions of "world view" and a language as well as the syntax of symbols are discussed in terms of their role in this effectiveness.

The final section "CHARACTER RECOGNIZERS" contains a general discussion of the structure of and capabilities desired in such systems which should be of fairly wide interest. Also included are technical descriptions of character recognition algorithms developed by Bernstein (Ref. 1) and Ledeen (Ref. 3). The former of these has been utilized in the pilot study which is described in APPENDIX A.

2.0 MAN-MACHINE INTERACTION

With the advances being made in time sharing and the low costs of display terminals, it is becoming practicable to permit easy access to computing machinery. However, easy access will go unused unless it is also possible for the user to communicate easily with the computer. A non-programmer with a problem needing solution should not find it necessary to become a programmer in order to find his solution. He needs a means which is both simple and natural * to communicate with the computer. The user does not want to be bounded in his abilities by fine details. The user prefers to utilize abbreviations and appropriate symbols. His language is context dependent. He does not want to have to specify that which appears obvious to him. For these reasons, the user would like a problem-oriented language; but this is not enough. The user wishes to introduce symbols and state initial parameters which belong to the particular problem on which he is working. As the user develops his solution, he develops a new language dependent upon the context of the problem. He names items, describes processes, and details properties. It would be impossible to keep ideas fresh or the user happy if he had to describe an item in detail every time it was referenced. A physicist would not want to write "Planck's constant," or the value of it, each time he referenced it. He prefers to use "h" instead.

2.1 History

In 1960, J. C. R. Licklider (Ref 4) prophesized "man-computer symbiosis." By this he meant the elimination of the use of the computer as a servant-tool and the creation of a partnership of man's random sort abilities with the computer's calculative abilities. The computer would become as much a part of man's thinking process as his legs are a part of his locomotive process. A man communicates with his legs by means of his nervous system, a communication channel readily accessible. Note that man has to become familiar with the means of communicating with his legs, requiring a great deal of time and concentration. Also that he is in almost constant contact with only one set of legs. But when it came to using an automobile to increase

* Natural here means something not too distant from the users range of familiarity. Nothing done with computers can be considered natural in the biologic sense.

his mobility, each man need spend only a small portion of time learning to control the automobile due to standardized, simple, and fairly natural means of control. No matter how cheaply or easily available an automobile, it would have very little use if each individual were required to be a mechanic having to design and install the steering apparatus each time he wished to use the vehicle.

2.2 Partnership of Computer

The state of computing currently requires this and as a result computers will not be utilized as they should be until the user no longer needs to be a programmer. Only by setting up a simple communication medium can this result be obtained. The means of communication is dependent upon man's thinking processes and the machine's calculating processes.

A computer's calculating processes handle data which is essentially two-dimensional. That is, its variables are time and core location. However, human thought processes are capable of operating on multi-dimensional variables, with coordinates such as height, width, depth, and time. The time coordinates can be simply identified. However, the translation of the spatial coordinates into core locations is non-trivial and requires analysis and programming.

2.3 Displays as Interaction I/O

Fortunately, two-dimensional spatially oriented data seems to be sufficient for most human thought processes. Either by prior training or by natural inclination, most people seem to be able to work with two-dimensional representations for problem solutions. This ability reduces a major portion of the communication problem to a translation of two-dimensions to one-dimension.

Display equipment provides the linkage between core locations and a 2-D working surface. Through input devices, such as the light pen or the data tablet, it is possible for the user to input data in two dimensions. The display screen allows the user to view output similarly. Although, not completely satisfactory, displays are more suitable for the representation of data in three or more dimensions than are other output media.

One of the problems in man-machine interaction is that the user tends to be bound to stop-and-go action. The lack of continuous action tends to be a perturbing influence to the user's line of thought. Currently, only displays allow the level of interaction that is necessary for man-machine symbiosis.

3.0 SEMANTICS AND SYNTAX

3.1 Computer Core Images as Artificial Representations of Real World

Computers do not contain real items in their memories. The contents of memory are artificial representations of some aspects of the real world. When the user communicates with a computer system, he is not communicating with the computer any more than he converses with the telephone during a telephone conversation. The computer is the tool which allows him the ability to communicate with an ephemeral object - a model of a portion of the real world. The computer is also the tool of the model. It is the means by which the model updates itself, and by which the model communicates with the user.

3.2 Semantics as Relationship Between Symbols and Real World

Communication is the passing of information from one source to another. Inherent in the passing of information is the requirement that the receiving entity have a potential to understand the information being transmitted. That is, the entity receiving the information must be capable of assigning a meaning to the symbol which was transmitted. The requirement is known as the semantic portion of communication. Semantics is the meaning of a communication.

Consider a blind man. To him the word "red" is a useless qualifier. Not having seen color it is impossible for him to assign a meaning to the word. He has no potential to understand the word as a specific color. Indeed, the abstraction represented by the word "color" has no meaning to him.

Communication can be successful even when the entities in communication have neither common objects nor common relationships between the objects. The potential to understand inputted signals does not necessarily imply that the receiving entity is familiar with the same objects and relationships that the sending object is familiar with. All that is required is that there exist a structure known to the receiving entity which is sufficiently similar to the structure of the sending entity. Such a similarity can be called an isomorphism between world-views for purposes of definition.

A "world-view", W, may be defined as:

1. A set, S, of objects; a, b, c, . . .
2. A set, U, of subsets of S; z, y, x, . . .
3. A set, R, of relations; m, n, . . .
   of the objects of S union U.

A relationship, m, of the objects, a and b, may be represented by m(a;b)

An isomorphic transformation, h, between two world-views W and W' is defined as a one-to-one correspondence of the objects of W' such that:

1. For all $a \in z$ and $z \in U$, there exists $z' \in U'$ such that $h(a) \in z'$ and for all $a \notin z$, then $h(a) \in z'$; i. e., $h(z) = z'$.
2. For all $m \in R$, where $m(a;b)$ exists; $a, b \in S$, there exists $m' \in R'$ such that $m'(h(a); h(b))$ exists, i. e., $h(m) = m'$.
3. There exists h' such that:

   i) $h'(h(a)) = a$; for all $a \in S$

   ii) $h'(h(z)) = z$; for all $z \in U$

   iii) $h'(h(m)) = m$; for all $m \in R$

If there is an isomorphic transformation between two world-views, these world views are isomorphic and a valid communication can exist between them. A language, L, can be formalized as a triple, (T, P, S), where:

T is a set of distinct symbols called the terminal set of L.

P is a set of procedures or syntax statements specifying permissible sequences or concatenations of the elements of T. P can be considered the grammer of L.

S is a set of concatenations of the elements of T according to the rules of P which are recognized as meaningful. S is called the set of sentences of L.

A communicative language between two world-views, W and W', is a language such that:

1. For each pair (k,k') where $k \in S$, $k' \in S'$ or $k \in U$, $k' \in U'$, or $k \in R$, $k' \in R'$, and $h(k) = k'$, $h(k') = k$ - there exists a distinct symbol, of L or a distinct sequence of symbols constructed according to the rules of P which may be associated with the given pair.

Now suppose that an entity with world-view, W, desires to communicate with a second entity with world-view, W'. The first entity wishes to make a statement equivalent to m(a;b). Using the proper communicative language, the first entity can make such a statement, which is interpreted by the second entity as m'(a';b'). This is a valid statement and has meaning in the second entity's world-view. The second entity can "reply" with the statement equivalent to n'(a';b'). Interpreting this to be n(a;b), the first entity has established a valid communication. It is inconsequential to the first entity that the second has interpreted m(a;b) as m'(a';b'). The relation n(a;b) may not have been initially apparent to the first entity, but since the world-views are isomorphic the relationship holds and the first entity has learned a new fact. Thus, as long as world-views are isomorphic, it is possible for the two entities to communicate successfully and usefully.

An isomorphic world-view provides a receiving entity with the potential to understand incoming signals. As such the entity is semantically capable of communicating.

Once such a potential to assign a meaning has been established, the symbol used to indicate the meaning becomes arbitrary. Suppose it was decided to utilize the word "kludge" as the symbol for computer. Such terms as "kludge language", "kludge processing", and "kludge error" become readily understandable. A number of children's games are based upon the substitution of strange words for familiar words. Teenagers become quite adept in the arbitrary assignment of meaning to words, often resulting in lack of communication with older generations.

This illustrates another condition to successful communication. Although the choice of symbols is arbitrary, the entities in communication must assign equivalent meanings to the symbols employed. In other words, definition of terms is necessary.

3.3 Syntax as Valid Relationships Between Symbols

As communication becomes more complex, simple definition of terms is not sufficient. Connections and sequences of terms are used to transmit varied meanings. Rules regarding the structure of these sequences are necessary to separate one meaning from another and sensible strings from nonsensible. For example, the sentences:

i) The boy bit the apple.
ii) The apple bit the boy.
iii) Apple boy the the bit.

are interpreted differently. Sentence i) has an obvious meaning. Sentence ii), while not representing any probable event, still contains meaning because the structure of the sentence is acceptable. However, sentence iii) is pure nonsense. Each of the words of sentence iii) has meaning individually, but the structure is unacceptable so there is no information transferred.

Thus, for complex communication to be successful, it is not only necessary to define terms but it is also required to define acceptable structures of term sequences (syntax). The rules and conventions used in the transmitting information constitute a language.

Such a language L can be simple or complex. A simple language would be one where the set of sentences is equal to the terminal set and the set of procedures is the null set. An example of such a language for man-machine communication would be the case where man at various occasions determines the action of the system by inputting a control character. The machine need not check syntax. It only needs to determine if the character is a member of the terminal set. If not, an error has been made and a default option should be taken.

A more complex language is required when a large number of distinct actions can occur as a result of the combination of many options. In such a case, some options may be legal only when specified in connection with a different option; some options may not be legal when specified with other options; the sequence of options may be important. The above conditions would call for a large and intricate syntax. The set of sentences would be efficiently described only by the syntax and would be much larger than the terminal set.

A language performs the function of determining symbols and establishing conventions regarding the use of the symbols for successful communications. Formalization of the language allows the designer of a system using that language to implement the interfacing smoothly. Also documentation becomes more straightforward.

Communication between entities may depend on more than one level of semantics. When a user communicates with a computer through character and syntax recognizers, semantics is involved on two levels. On one level, the semantic is the character. The strokes represent symbols and their relationship to each other is the syntax. If the symbols and their relationships are proper then it is possible to recognize the character. The characters then become symbols for the next level of communication. If the relationships between the symbols is in the proper syntax, then meaning can be associated to the sequence of characters. In this way more varied and meaningful information can be transmitted.

In a man-oriented communication system, it will be necessary to use a hierarchy of syntax where communication is really on a number of levels. Parallel systems of semantics and syntax will be required. .This would be necessary to introduce meta-languages, where the user is able to construct or modify his existing language. It is a well known fact that humans think in modular fashion. A given sequence of symbols with a given syntax in different modes can have entirely different meanings. The statement "There are a lot of bugs!" has different meanings depending on whether you are talking to an exterminator or a programmer. These modes must be set previous to the statement by some kind of higher language. The higher language may not necessarily be explained as a higher language. It could be in the form of an orientation in the conversation. This is called context-dependent. Initially, it would be expected that context- dependent languages for man-computer conversation would be gross in mode setting parameters. That is, they would utilize explicit symbols and mechanizations to switch from one language to another. The hierarchical interpretation of languages would not cause such a problem since each sublanguage would indicate how well it was being input. A character recognizer would replace the tracking points with a neat character to indicate that it has interpreted the input correctly. The user can make corrections and modify his character set at that level without interrupting the higher language. Figure 1. is an example of a hierarchical language system for syntax recognition with the ability to modify syntax.

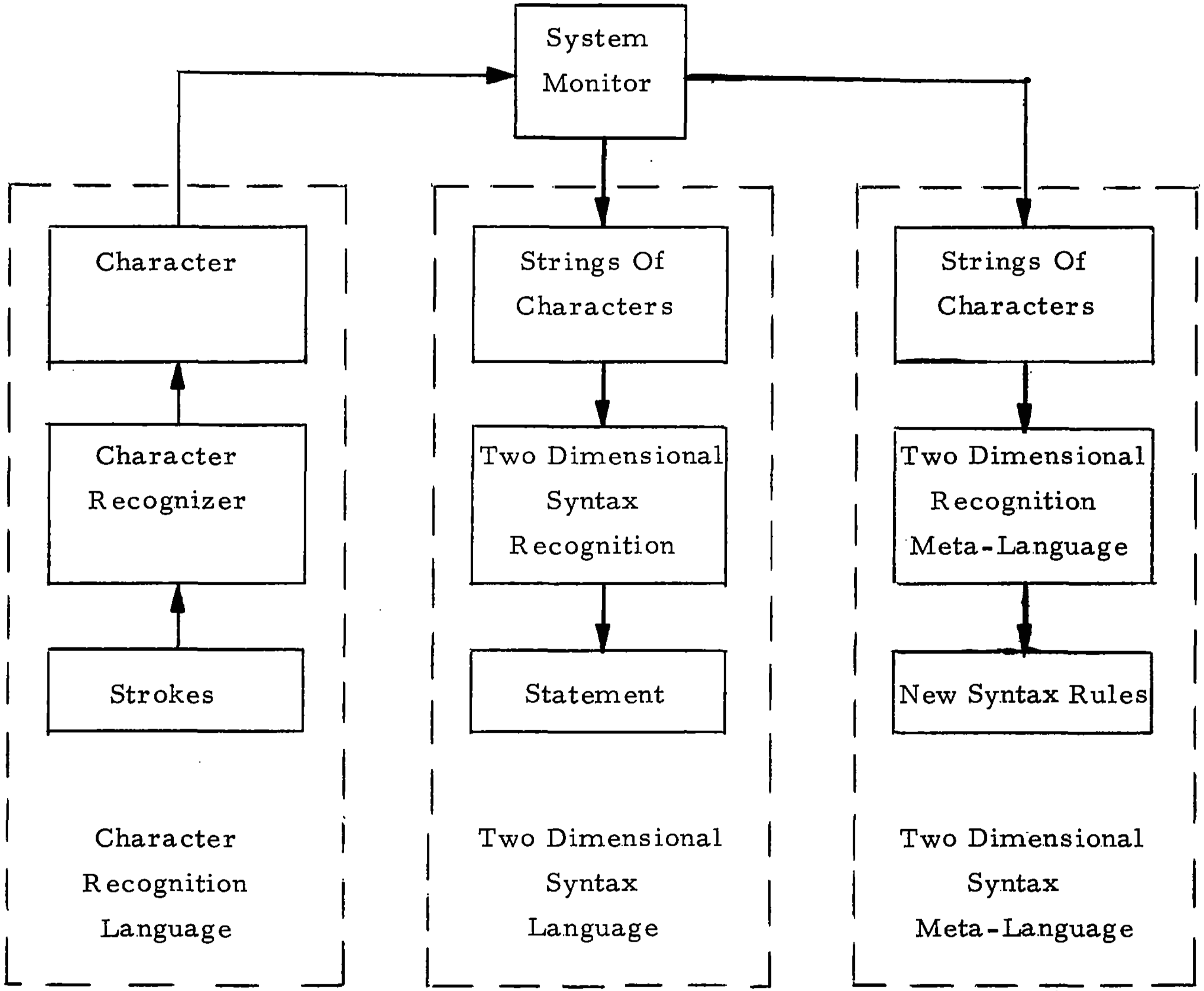

System Monitor
Character
Character Recognizer
Strokes
Character Recognition Language
Strings Of Characters
Two Dimensional Syntax Recognition
Statement
Two Dimensional Syntax Language
Strings Of Characters
Two Dimensional Recognition Meta-Language
New Syntax Rules
Two Dimensional Syntax Meta-Language

FIGURE 1

Example of Hierarchical Language System

4.0 CHARACTER RECOGNIZERS

4.1 Need of Character Recognizer for Interaction

The principle rationale behind the development of graphical display consoles has been the desire to provide a tool which would allow individuals to communicate with a computer in a concise and natural manner. An interactive display should allow the user to input graphically as well as receiving output graphically. In order to meet these demands, light pen tracking routines have been devised and tablet input devices have been invented. In the realm of text input, the teletype would appear to be supreme. However, in certain areas it is both difficult and inefficient. These areas are primarily where the user inputs text infrequently such as in the labeling of drawings or where the position of the text is not linear such as in complex mathematical expressions. In order to facilitate the input of text under these conditions, various character recognizers have been experimented with.

4.2 Adaptive Symbol Generation

A proper character recognizer should contain the ability to adapt to various input. Very few people write or print in the same fashion. Complex characters become almost individual in composition. As a result, character recognizers must be designed so that they adapt to the various individual inputs or people must be constructed so that they adapt to the format specified by the computer. Obviously, the first alternative is the only feasible one.

A proper character recognizer should also allow the on-line construction of new and non-standard characters at the user's option. Persons who have used typewriters are familiar with the feeling of loss when they are unable to use a desired character. Recent advances allow the change of the characters on the typewriter, but this requires manual intervention and is inefficient when one must switch between alternate character sets. There is the associated problem of what to do when the character you wish to use is just not available on any of the character sets. To eliminate this problem, the character recognizer implemented on the CDC3200-IDIIOM configuration was modified to allow the on-line creation of characters by the user.

4.2.1 Character Semantics. - A character to a person is a two-dimensional figure of a given general shape, the size is irrevelant. A person associates a character with a given sound or as a particle of another entity called a word. Also, the character may be considered as a symbol representing some object or state in a person's world-view. In either case, the character is not considered as an object itself, but rather a representation of some item or event. To a computer, a character is a given sequence of bit settings in a memory register. Simply, it can be referred to as a character code according to some predefined coding scheme such as ASCII or EBDIC. It is important to note here that the code assigned to a given character is immaterial. It is necessary only that the code be assigned and that the logic which handles characters is consistent with the given coding scheme.

In character recognition, the semantic of a character is the sound or symbol associated with it. Within the computer the semantic is the character code. The difference in these semantics is of an order of magnitude. The character code recognized by the computer is only equivalent to the human recognition on the shape level. The equivalency of these levels can be shown by the recognizer's ability to call upon a routine which will display the proper character for a given character code. The computer does not associate any meaning to the character code, at least not at the recognition stage. The isomorphism between the computer's world-view and a user's can be extended as is done with the syntax analyzer.

Character Syntax. - Character recognition may be depicted as an application of a syntactic language, since an inputted character is usually broken down into a "sentence" of features and these features are "parsed" to determine if they are meaningful sentences.

An inputted character consists of a series of strokes. Each of these strokes may have different forms; they may be horizontal, vertical, diagonal, or curved. Also, the relationship of the strokes to each other is important. The relative size of a curved stroke provides the difference between a P and a D. These forms and relationships are known as the features of the character. A character recognizer initially processes the input strokes to determine the features. These features are coded and set in a series known as a feature string.

The coded features may be considered the terminal set used by the language processor, (the character recognizer). The feature string is considered a sentence. It is necessary to determine whether the sentence is meaningful; that is, does the feature string represent a real character. This is done by comparing the feature string constructed from the input strokes with the feature string used to define a character. The feature strings which define the characters are the syntax rules of this language. An example of a character recognizer as a language processor is illustrated in Figure 2.

The character A is described by the feature string 1 2 3. Such an inputted feature string would be parsed as < partial A > (substring 1 2), then parsed as <A > (string of form <partial A > 3). Since < A > is an element of S, the set of meaningful sentences, the character would be recognized. Parsing would be done by matching segments of the inputted feature string against a tree of feature strings. Such a tree is another representation of the syntax rules of the language. (See Figure 3). If the inputted feature string can follow such a tree and end at a node which is in the set of sentences, S, then it is a recognizable character and can be assigned the character code associated with that node.

The use of a tree structure to represent the syntax rules of a language makes it possible to easily change and add to the syntax rules; thus modifying the language. By utilizing a character recognizing language, and by programming general purpose routines to manipulate and modify the tree which is used to specify the syntax rules for the language, it is possible to construct an adaptive character recognizer. Such a character recognizer would "learn" to recognize characters and, for that reason, would not be restricted to a small set of given character constructions, but would adapt to the style of the individual using the recognizer.

Initially, the tree would contain a single node, that of the null string. A character is drawn and its feature string is generated. Since the tree is essentially empty, the character cannot be recognized. The character recognizer indicates this in some manner such as displaying the set of characters it is capable of recognizing * (See Appendix A.) By indicating that it cannot

* It is here (among other places) that the potentiality to understand becomes important. As impossible as explaining "red" (in the context of vision) to a blind man is the task of defining an unrecognized character to a system that does not have a code or the ability to generate a code to assign to the character.

Feature Coding:

1 = /

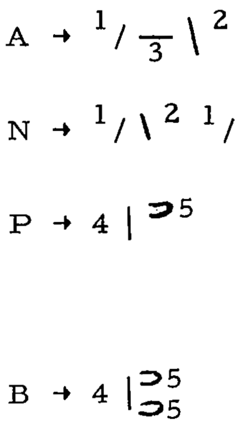
2 = \

3 = —

4 = |

5 = ⊃

Characters:



Terminal Set:

T = {1, 2, 3, 4, 5}

Syntax Rules:

<partial A> : = 1 2

<A> : = <partial A> 3

<N> : = <partial A> 1

<P> : = 4 5

<B> : = <P> 5

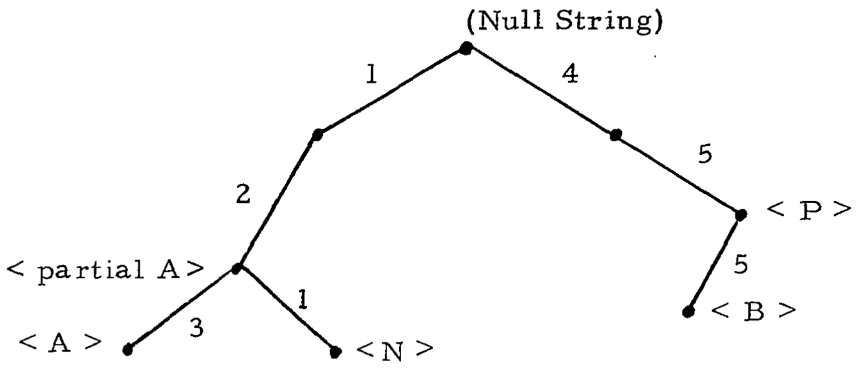
Set of Sentences:

S = <A>, <N>, <P>, <B>

Note: This example does not utilize or take into account the relationships of the strokes to each other.

FIGURE 2

A Simple Character Recognizing Language



The starting point represents the null string (no features). Second level nodes are not labeled as they do not represent syntactic constructs in the Simple Character Recognizing Language.

FIGURE 3

Representation of Syntax Rules for Simple Character Recognizing Language by a Tree recognize the character, the system is informing the user that according to its current syntax rules the input is not a "sentence" to the character recognizing language. The user now has the option of adding a syntax rule which will allow similar input to be interpreted as a sentence of the character recognizing language. This is done by adding the feature string to the tree (if necessary) and assigning a character code to the node reached by the feature string.

In this way the recognizer adapts to the user. The syntax rules which are used to recognize the characters are generated by the user's style of handwriting. It would be ultimately possible for the system to recognize user's handwriting without training if the memory could hold a large enough dictionary of feature strings and if the search through such a dictionary was quick enough to be feasible. The size of such a memory has not been determined and it remains unknown whether a system which would recognize 90% of the user's handwritten characters without individual training is practical or not.

Training for a character recognizer can be set up in two fashions. One is training during use. This technique was used in the pilot study. (See Appendix A). As the user communicates with the system, unrecognized characters are allowed to be added to the dictionary. Although this method would seem desirable, it interrupts the logical train of thought of the user who is forced to momentarily perform an unrelated task. The other method is to only allow training before use of the system. When the user has reached his desired level of recognition, he signals the end of the training session, and begins communicating with the system. During communication, the system indicates unrecognized symbols by a special code and allows the user to reconstruct the character immediately. Such a system would allow the user to re-enter the training mode whenever he felt the necessity.

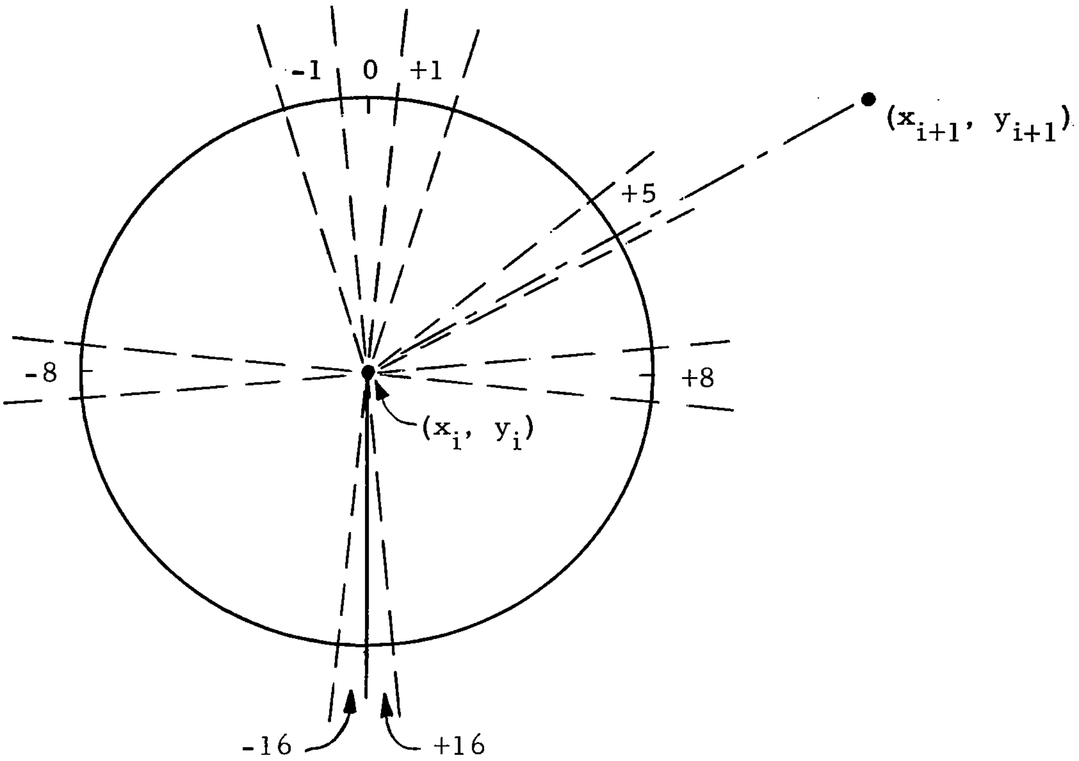
4.3 Character Recognizers

4.3.1 Bernstein. - The pilot study has utilized a character recognizer for hand-printed input developed by M. I. Bernstein (Ref. 1) which will be referred to as the BCR in what follows. The BCR consists (in a functional sense) of three sections: input preprocessing, stroke(s) feature extraction, and dictionary construction and searching. The preprocessing section separates the raw input data from the tablet into strokes, smoothes and filters this raw data, counts the points rejected by filtering, and provides immediate visual feedback to the user by a CRT display of the points corresponding to the smoothed and filtered data. The smoothing process employs an eight-point moving average of the raw data point and its seven predecessors. To start the process for a stroke, the first input point is replicated eight times. Filtering is applied to the smoothed data and is simply the comparison of absolute values of the differences between the last accepted point and the current point. If either $|\Delta x|$ or $|\Delta y|$ is greater than the filter constant (a program parameter), the point is accepted and displayed on the CRT; otherwise, it is rejected and the point count of the previous accepted point is incremented. These rejected point counts are a measure of the pen velocity (inverse relationship) and are used in corner detection.

The capability of scrubbing (erasure) of input is provided by sensing whether the smoothed and filtered data fills the input buffers (the pilot study currently has 200 word buffers.) If so, the input is considered to be a "scrub", and any displayed character(s) within the scrub range are erased.

The feature extraction portion of the BCR analyzes each stroke of the preprocessed input to determine both primary (path) and shape features, and generates an internal representation (a feature string) of concatenated codes for these features. The first step of this analysis is to convert the coordinate points of the stroke into discrete headings (See Figure 4), thus making the stroke position-independent. Adjacent headings are then differenced, and each difference is associated with the point common to the two headings. Next, the beginning and end of the stroke (the first and last heading differences) are tested against a threshold to determine if a "hook" exists. If the difference exceeds the threshold (the current value is 5) the hook is eliminated by removing the first and/or last point from the stroke.

With these preliminaries accomplished, attention is focused on the extraction of primary features, namely, corners and inflection points. With the exception of a few points at the beginning and end, data associated with each point of the stroke is examined. Following Bernstein's notation, let Δh_i denote the heading difference associated with point i. Also, denote the



Discrete heading associated with the point (x_i, y_i) is +5

FIGURE 4

Discrete Heading Values for Coordinate Points count of points rejected by the filtering process and associated with point i by c_i ; then, a point i is marked as a corner if

1. $|\Delta h_i| > 11$ (a "true" corner) <u>or</u>

2. $|\Delta h_i + \Delta h_{i+1}| > 13$ (a "split" corner) <u>or</u>

$$\left.\begin{array}{ll} \text{3.a.} & c_i \geq 8 \quad \underline{\text{or}} \\ \text{b.} & c_i + c_{i-1} \geq 14 \quad \underline{\text{and}} \\ & c_i + c_{i-1} > 6 \cdot \min(c_{i-2}, c_{i+2}) \end{array}\right\} \begin{array}{l} \text{(A "local velocity" or} \\ \text{"rejected point count"} \\ \text{corner)} \end{array}$$

Those cases for which there are two candidates (2 and 3b) are resolved by selecting as the corner the point with the largest $|\Delta h|$; point i is arbitrarily selected when $|\Delta h_i| = |\Delta h_{i+1}|$. It should be noted here that the existence of a corner constitutes a division of the stroke into sections from which shape features and inflection points are extracted separately. A stroke with no corners is processed as a unit for the purpose of this extraction.

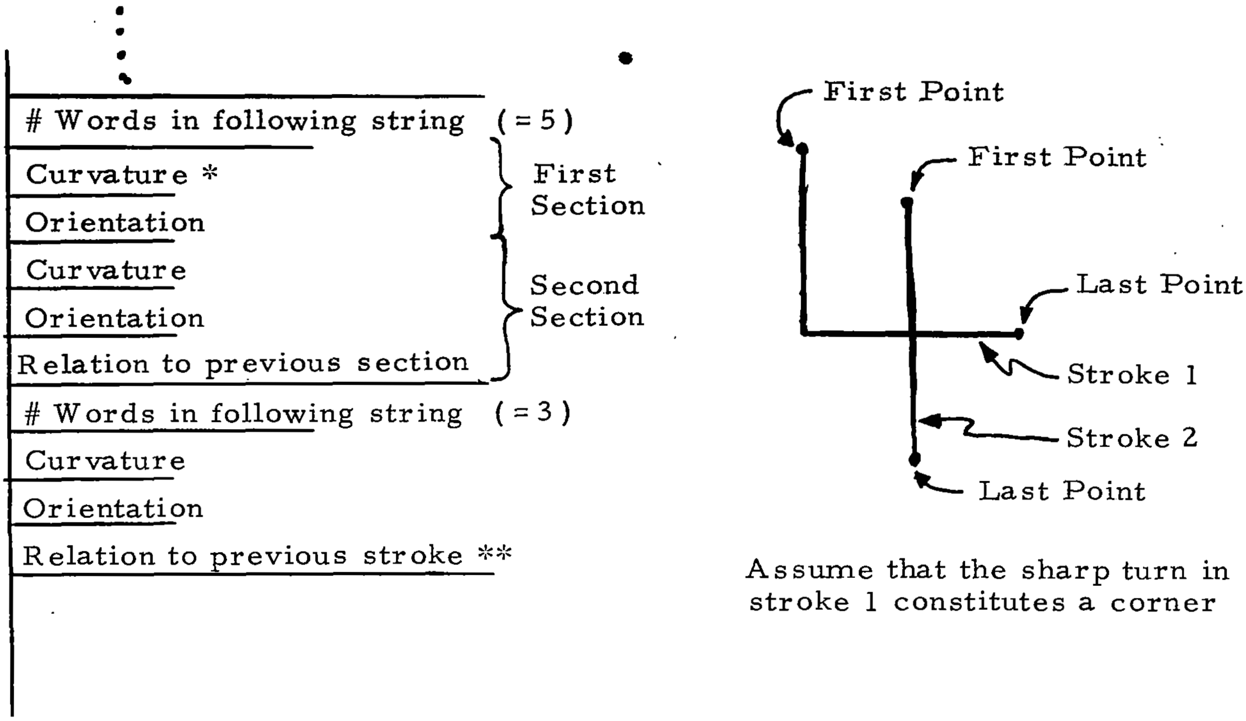
Inflection points are detected by examining the same heading differences utilized above. First, unintended inflections resulting from minor wiggles are excluded by requiring the absolute value of the sum of the Δh s for the stroke (or section thereof defined by a corner) to exceed a threshold value (presently equal to 5) prior to further examination. After the threshold is exceeded, differences between successive maximum and minimum values of the sum of Δh s are tested to see if they are > 6. If so, an inflection point exists either at the maximum/minimum or, if there are a series of points between the extreme values which form a straight line, at the midpoint of this straight line. The applicable point is then marked as an inflection point.

The shape features of interest for a stroke or section thereof are curvature, orientation, and geometric relation to the composite of previous sections of the stroke, if any. The section's curvature is simply the sum of it's heading differences - the code used in the feature string is determined by which of a series of curvature ranges the sum falls in (See Figure A-2, Appendix A.) The orientation extraction utilizes the direction between the first and last points of the stroke, the code being one of eight discrete heading values. Exception is taken where the section or stroke is essentially a closed figure, a case denoted by a code of zero. For strokes which have been sectioned by one or more corners, codes expressing the relation of the current section to the composite of previous section(s) are generated and placed in the stroke's feature string. This is accomplished by computing the rectangle surrounding the current section and the rectangle surrounding the composite of previous sections, the code being one of eight discrete heading values associated with the direction of the line joining the centers of the rectangles. Again, exception is taken when the rectangles are "coincident", a case denoted by a code of zero. Finally, the existance of a section which is defined by a corner is reflected in the feature string by changing the sign of it's curvature code.

The generation of a feature string for a stroke is not sufficient for the description of multi-stroke characters. Consequently, further analysis and extraction relating (via additional codes in the stroke's feature string) the stroke to the composite of previous strokes of the character is required. A portion of this analysis is accomplished in the same fashion as the determination of the relation of sections of a stroke to the composite of previous sections, namely, generation of a code equal to zero if the stroke is coincident or equal to one of eight heading values if not. Further, for non-coincident strokes, this code reflects whether the current stroke is "near" or "far" from the composite of previous strokes.

At this point, the BCR has processed the input to the point of having generated and stored it's internal representation. An example of these feature strings is shown in Figure 5.

Having generated the feature strings representing an input character, the BCR next turns to the central task of character recognition which is performed by the dictionary construction and searching portion of the algorithm. On a stroke by stroke basis, the dictionary is searched for a match between existing (previously generated) feature strings and the string representing the stroke under consideration. If the strings for each stroke match and the



# Words in following string (= 5)

Curvature *

Orientation

Curvature

Orientation

Relation to previous section

First Section

Second Section

# Words in following string (= 3)

Curvature

Orientation

Relation to previous stroke **

Assume that the sharp turn in stroke 1 constitutes a corner

* Curvature code is negative because first section of first stroke is defined by a corner.

** This code also reflects, for non-coincident strokes, whether the stroke is "near" or "far" from the composite of previous strokes of the character. In this example, however, this additional information would most likely not be present, since stroke 2 appears to be coincident with stroke 1.

FIGURE 5

Feature Strings for the Input Character "4" dictionary's pointers to the existing feature strings in fact point to these matched strings, the character is recognized, the appropriate character code is returned, and a "clean" version of the character replaces the handdrawn input on the display. If the character is not recognized, pointers are set for subsequent entry of the new string into the dictionary. Also, the standard character set (See Figure A-3, Appendix A) is displayed so that the user may specify the symbol that the input was intended to be. As an alternative, the user may input a scrub, in which case the undefined input is erased from the display, it's associated data deleted from memory, and the BCR awaits the input of another character.

If the user specifies the intended symbol, the appropriate character code is supplied to the dictionary entry and the entry's feature string pointer is set to the string(s) corresponding to the input. Finally, a "clean" version of the specified symbol replaces the handdrawn input on the display.

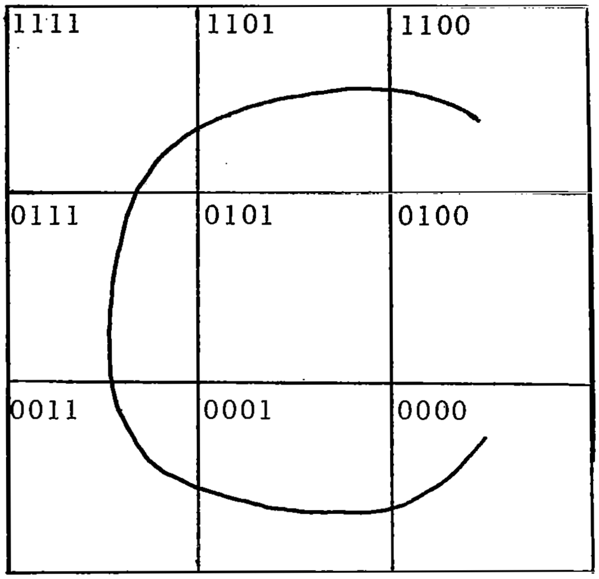
4.3.2 Ledeen. - The Harvard University Character Recognizer (HUCR) was developed by Kenneth Ledeen for his doctoral thesis. It is based on the principles contained in Teitelman's paper on character recognizers in that it normalizes a rectangle containing the character input and builds feature strings according to stroke direction, position, and form within the normalized rectangle (Ref. 3). The form of a stroke in the HUCR becomes implicit in the description of the direction and position.

The HUCR consists of three sections: input handling, property extraction, and recognition procedure. The input handler monitors the incoming points from the tablet and provides feedback by a point display on a CRT. Using a real-time clock, the velocity and position of the pen is determinable. The points are separated into strokes and the minimum and maximum values of the x and y positions are recorded for future normalizing.

The property extraction computes the range of the stroke and determines if the stroke was a dot, horizontal, or vertical. If the stroke is not one of the above special cases, the stroke is normalized into a three by three rectangle and a property list of four vectors for the stroke is generated according to the sections which the stroke passes through. (See Figure 6). The four bits, used to encode each of the nine smaller rectangles (See Figure 6a) are used in the construction of the four property vectors. The

1111	1101	1100
0111	0101	0100
0011	0001	0000

a) Normalized and Coded Rectangle



b) Stroke for Character "C"

c) Property List for Character "C":

1100
1101
1111
0111
0011
0001
0000

d) Property List as Vectors

I:	1110000
II:	1111000
III:	0011100
IV:	0111110

e) Condensed Vectors:

I:	10
II:	10
III:	010
IV:	010

FIGURE 6

HUCR Property Extraction

four property vectors are stored in a 4 x n array (where n is the number of strokes.) Also the x, y midpoint of the stroke is recorded for later use in a pseudo-stroke.

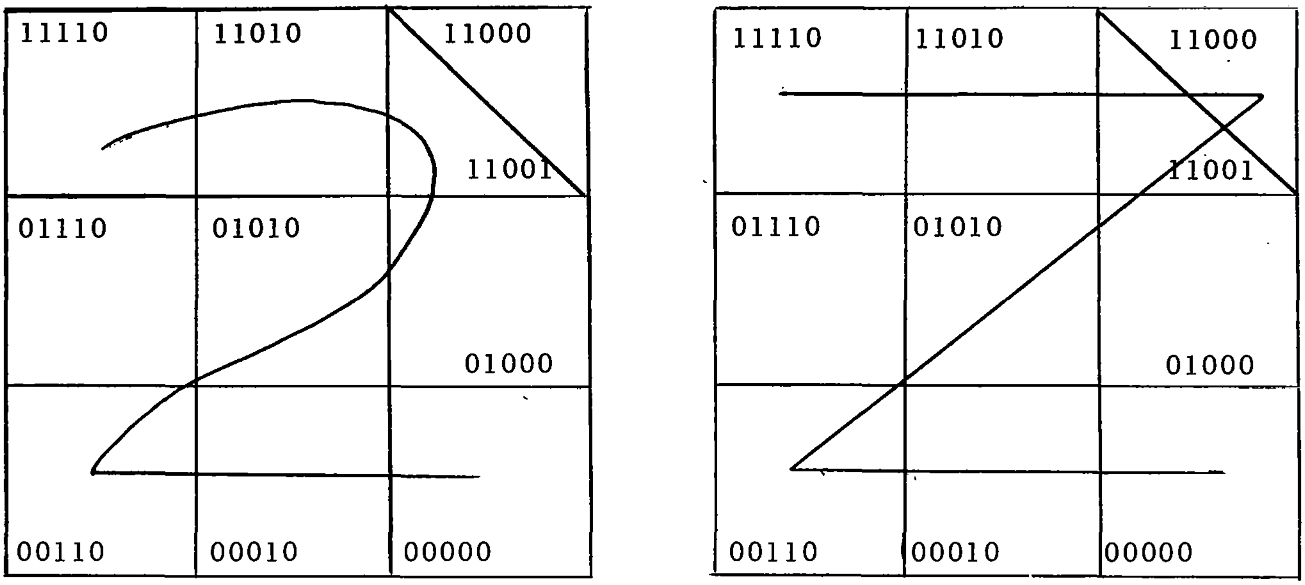
After the property vectors for all the strokes have been generated, a final set of property vectors is generated for the pseudo-stroke consisting of the midpoint values with the character ranges as the stroke dimension. The psuedo-stroke is used to describe the interrelationship between strokes.

At this point the property vectors for all the strokes are concatenated, using a special symbol, w, to denote the end of a stroke. This produces four "super" property vectors which completely describe the character.

The recognition procedure correlates the input character matrix with a character dictionary. The dictionary is a tree structure with four sections, one for each property vector. The property vector represents a path through the tree. At the end of this path is a sublist of "candidates" or character codes and a weight assigned to each. The recognizer merges the candidates and their weights from each of the four sections of the dictionary. The character with the highest weight is returned as the recognized character. The weights may be altered after each recognition and provides the recognizer with a "score" on how it is doing.

The HUCR has the property of allowing the user to define sections of special interest for recognition purposes in cases where the four initial property vectors are not sufficient to determine the character inputted. The nine inner rectangles can be sub-divided into smaller sections and a fifth or higher order bit is added to the section code. Also a new property vector is allowed for and a new section of the character dictionary may be generated. With this property, the user may distinguish between such similar characters as 2 and Z. (See Figure 7). The importance of the property vectors can be adaptive also in that they may be weighted by use and periodically, the order in which they are examined during recognition may be updated.

The HUCR, due to the fact that the property vectors consist of binary bits, is fairly compact. The dictionary can be rather tight in that a condensed property vector is only a string of alternating 1's, 0's, and w's and can be implicitely represented by position in the tree structure. (See Figures 8 and 9). Another factor which allows compactness is that using the



a) Character "2" with Added Region

Property Vectors:

I:	10
II:	10
III:	1010
IV:	1010
V:	010

b) Character " Z " with Added Region

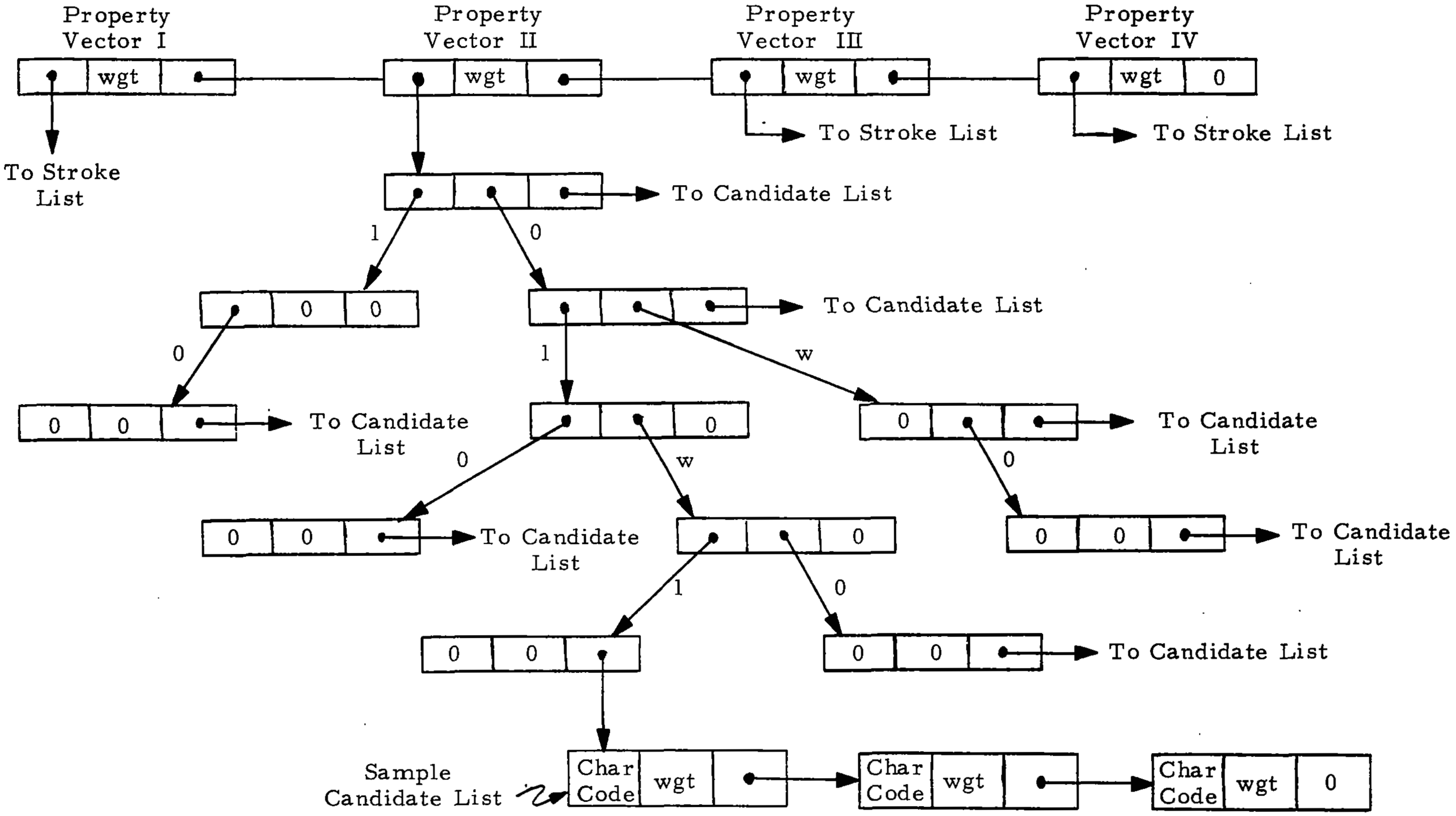
Property Vectors:

I:	10
II:	10
III:	1010
IV:	1010
V:	01010

NOTE: The only difference in the property vector description of "2" and "Z" occurs in the added vector, V.

FIGURE 7

Additional Region for Improved Discrimination



NOTE: Zero in a section of "word" indicates end of tree branch.

FIGURE 8

Dictionary Tree for HUCR

Previous Pointer Represented	Left Pointer Represents	Right Pointer Represents
1	0	w
0	1	w
w	1	0

FIGURE 9

Pointer Positioning for HUCR Dictionary Tree

HUCR's property vector scheme, each character has only a few different descriptions possible. These features and the lack of dependency on arithmetically-based features allows the HUCR to be economically coded in assembly language and suitable for use in small machines.

4.3.3 Relationships and Differences. - A comparison of Bernstein's and Ledeen's character recognizers produces some interesting relationships and differences. First, each recognizer performs it's job via the same three general processes, i.e., one that converts the real world into data readily amenable to analysis, a second which provides an internal representation of the properties/features of the input characters, and finally, a recognition process which adapts itself to the user's penmanship idiosyncrasies. However, the "training" required to accomplish this adaptation is performed in different fashions. As was mentioned previously, the pilot study (the BCR) is trained on a continuous basis during use, i.e., the feature strings of each unrecognized character and the character code specified by the user are added to the dictionary, thus updating the BCR's recognition ability. On the other hand, the HUCR allows training only before actual use of the system.

Each of these training schemes has it's advantages. Training previous to system use allows (1) subsequent uninterrupted input by the user (except for re-entering of unrecognized characters) and (2) control over limitation of dictionary size, but creates the problem of leaving the user in doubt as to how much training is sufficient since he/she receives no feedback re recognition performance. Continuous training provides the concomitant benefit of performance feedback, but interrupts input and can lead to a space consuming and unwieldly dictionary.

The relative efficacy (with respect to accurate recognition) of the property/feature coding remains a moot point. Comparison of the character's internal representations alone gives the intuitive impression that Ledeen's method (HUCR) is superior, at least when used in the recognition of alphameric characters. However, Bernstein's approach (BCR) coupled with continuous training has provided adequate recognition results, notwithstanding the resulting dictionary size. It is obvious that improvement can be made by combining the qualities of each method.

The philosophies inherent in the recognition portions of the HUCR and BCR show a basic difference. The BCR is deterministic, i.e., recognition requires an exact match of feature strings in it's dictionary and allows no "guessing." Further, no manipulation of the dictionary for the purpose of improving recognition ability is accomplished during use of the system. On the other hand, the HUCR has a quasi-probabilistic nature. The assignment of weights to the character codes in each of the four candidate lists enables the recognizer to select the character which is "most likely" the user's intention. Also, the ability to alter the weights provides further adaptive nature to the recognition process.

This type of approach can be incorporated into the BCR. One method, albeit of remote connection, is to restrict the addition of unrecognized characters' feature strings to the dictionary. Specifically, only a certain number of sets of strings representing each character would be allowed, this number perhaps being dependent on the inability of the system to recognize the particular character. After reaching the limit, this restriction could be maintained by replacing the oldest set of strings by the most recent, and this limit could be reduced as the system's recognition ability (for that character) improves. This approach would provide an adaptive nature in the sense that a user's input of any particular character may approach a constant form and thereby requires less allowance for variation within his penmanship (as opposed to variations between users.)

A final comment concerning the relative amount of coding required for implementation of these character recognizers. Although it is suspected that the BCR requires more space, the pilot study was approached from the viewpoint of ease of implementation rather than economy of size, since the GSFC CDC 3200-IDIIOM system imposed little restriction with regard to core memory usage. Also, the BCR was in part available in FORTRAN for an IBM computer and thereby reduced the effort required to code the algorithms for the CDC computer.

CONCLUSIONS

A conceptual framework for man-machine communication has been described, and technical feasibility has been demonstrated for an adaptive character recognizer which accommodates special symbols in addition to its regular alphanumeric set of recognized characters. However, the pilot model of the recognizer exhibits several significant problems in respect to practicality. These problems must be solved in order to progress from character recognition to the recognition of mathematical expressions and thence to a practical tool for the physical scientist. They are as follows:

1. The problem of computer memory size requirements. The recognizer as implemented used all 16K of the core memory of the GSFC CDC-3200 computer. This left no room for constructing a math recognizer around the character recognizer. Some savings might be realized by converting the program from FORTRAN to more efficient code, but the bulk of storage is required by data. The adaptive algorithm is in fact capable of using storage without limit.

2. Instances of recognition ambiguity. Pilot testing of the technique revealed difficulty of discrimination between "S" and "3", for example. This is believed to be a systematic error which may be resolved by additional analysis procedures. Another type of ambiguity may be predicted, in which identical patterns eventually occur representing similar characters such as "B" and "8", and which might be resolved by procedures which detect such instances as they occur. Recovery from either type error would require additional programming.

3. Human factors aspect of the physical embodiment. The present embodiment of the technique includes equipment and programs associated with a centrally located medium-scale computer. Since a computer must be dedicated to just the servicing of one user at a time, it makes sense for the user to come to the computer. It is an accepted tenet of human factors engineering

that this type of user (the physical scientist) will not generally accept the idea of learning to use a new tool such as this in public view, and beyond this obstacle it may be most inconvenient for him to bring all of his references and working paper to a central site. Therefore, embodiment in a small, inexpensive desk-top unit is indicated.

4. Cost/effectiveness aspect of the physical embodiment. It would not seem feasible to place a CDC 3200, with over 16K of core, on or near the desk-top of each physical scientist.

Because of these problems we conclude that further research should be aimed at the development of a minicomputer-based vehicle for graphical input tablet servicing, character and math recognizer implementation, or other means by which the physical scientist might intimately control the complex data processing operations involved in the analysis and meaningful interpretation of large volumes of sensor data from scientific spacecraft.

REFERENCES

1. Bernstein, M. I., "Hand-printed Input for On-Line Systems," System Development Corporation TM - 3937/0000/00, April 14. 1968.

2. Grover, G. F., "Real-Time Recognition of Hand-Printed Text," Proceedings of Fall Joint Computer Conference 1966, pp. 591-601.

3. Ledeen, Kenneth, "Harvard University Character Recognizer Technical Description," Unpublished draft of Doctoral Thesis, 1968.

4. Licklider, J. C. R., "Man Computer Symbiosis," IRE Transactions on Human Factors in Electronics, HFE-1, pp4-11, March 1960.

5. Miller, George M., "On-Line Computer Recognition of Handwritten Symbols," Electrical Engineering Department, University of Wisconsin, PhD Dissertation, 1969.

APPENDIX A. THE CHARACTER RECOGNITION AND CONSTRUCTION PROGRAM

Introduction

A brief description of the pilot study implemented on the CDC 3200-IDIIOM system at the GSFC installation (See Figure A-1) is provided by the following overall flowcharts and figures. The description of the Bernstein character recognizer (incorporated in the program as subroutines LISTEN, ASTROKE, and DICT) is in the section of this report entitled, Character Recognizers, and is not repeated here. Charts A-1 - A-4 show the overall flow of the main control routine CHARREC which resides in the CDC 3200. The flowchart includes the calls to the three subroutines listed above, the subroutine V620 which transmits function code and character code/size/position data to the Varian 620/i, and character construction subroutine CONCHAR.

Routine CHAR620

The IDIIOM display is controlled by a program (CHAR620) residing in the Varian 620/i. When CHARREC routine V620 sends function code data, the Varian 620/i Interface Control Unit causes an interrupt at a specific location which contains a jump to instructions which in turn route the flow to the subroutine which accomplishes the task(s) specified by the function code. A description of these eleven subroutines is given in the following subparagraphs.

ADPT. - ADPT (function code = 0) adds a single point to the tracking point buffer. The x-position of the point is read in, combined with the position x register command and inserted into the tracking point buffer. The y-position of the point is read in next, combined with the position y register command and inserted into the tracking point buffer. Finally, a short point command is inserted into the tracking point buffer, which displays the point at the given location.

ETPB. - ETPB (function code = 1) erases all the points from the tracking point buffer. The pointer to the tracking point buffer (TPTR) is reset to the beginning of the buffer, and the IDIIOM no data code is inserted into each of the 600 locations.

DCHA. - DCHA (function code = 2) displays a "neat" character in the location where the input character was drawn. The ASCII character code is read in from the CDC 3200, along with the XMIN, XMAX, YMIN, and YMAX values. The size of the character is calculated by subtracing XMIN from XMAX and YMIN from YMAX. The pointer to the character buffer is reset to the beginning of the buffer and a search is made to find a blank spot in the character buffer. When a blank spot is found, the XMIN value is combined with the IDIIOM position x register command and inserted into the character buffer, and the YMIN value is combined with the IDIIOM position y register command and inserted into the character buffer. Then the symbol dictionary is searched for the matching ASCII code. In addition to containing the ASCII code for all characters in the character set, the symbol dictionary contains points to the display commands and the extent of the characters in the character set. The size of the displayed character is divided by the extent of the character in the character set to determine the scale factor in both x and y. The display command that is pointed to by the proper pointer in the symbol dictionary is examined to determine if it is a x command or a y command. Bits 0 through 10 are interpreted as a delta x if bit 11 is a one and as a delta y value if bit 11 is a zero. Delta x (delta y) is multiplied by the scale factor in x (y) to determine the new delta x (delta y). The new delta x (delta y) is combined with bits 11 through 15 of the display command in the character set and inserted into the symbol buffer to give the new display command. If bit 15 of the next command in the character set is a zero, the command is processed as above. However, if bit 15 is a one, it is a position command for the next character and a jump and mark to symbol buffer instruction is inserted into the character buffer following the position y register command.

ECHA. - ECHA (function code = 3) erases a given displayed character. The ASCII code for the character is read in along with its XMIN, YMIN position. The XMIN value is combined with the IDIIOM position x command and the YMIN value is combined with the position y command. A search is made for matching values for both XMIN and YMIN. When the character is found, the position x and position y command, along with the jump and mark to symbol buffer instruction is erased from the character set.

Because the display commands in the symbol buffer are of varying length, depending upon the complexity of the character or symbol, they are not erased and the symbol buffer may overflow before the character buffer is filled.

EACH. - EACH (function code = 4) erases all the character buffer and symbol buffer. The pointer to the character buffer (CPTR) is reset to the beginning of the buffer and the IDIIOM no data code is inserted into each of the 200 locations. Then the symbol buffer pointer is reset to the begin- and the IDIIOM no data code is inserted into each of the 500 locations.

DCHS. - DCHS (function code = 5) displays the character set by linking the tracking point buffer to the character buffer. This is accomplished by changing the contents of location TPBF + 602 as shown in Figure .

ECHS. - ECHS (function code = 6) restores the linkage between the character buffer and the tracking point buffer. This is accomplished by changing the contents of location TPBF + 602 as shown in Figure .

DOVR (function code = 7) displays the overflow error message "DICT FULL". This is accomplished by changing the contents of location TPBF + 602 as shown in Figure .

TRAN. - TRAN (function code = 10_8) transfers the display file of a new symbol (created on line) from the CDC 3200 to the character set. The ASCII code for the new symbol is read in and inserted into the dictionary. The location for the new symbol is determined and added to the character set. Next, the pointer to the new symbol in the character set is inserted into the dictionary and the scaled down size of the new symbol is read in and inserted into the dictionary. Finally, the display commands are read in and inserted into the character set. The command 177777_8 signifies the end of the display file.

TEMP. - TEMP (function code = 11_8) clears the temporary buffer and links it to the tracking point buffer. The IDIIOM no data code is inserted into each of the 30 locations in the temporary buffer and the buffer is linked to the tracking point buffer by changing the contents of location TPBF + 602 as shown.

DLIN. - DLIN (function code = 12_8) inserts into the temporary buffer the display commands for a single line of the symbol being constructed. The ASCII code for the line (horizontal, vertical, oblique position or negative line) is read in along with the XMIN, XMAX, YMIN, YMAX values. The XMIN and YMIN values are combined with the IDIIOM position commands and inserted into the temporary buffer. Next, the display commands necessary to draw the line are inserted into the temporary buffer.

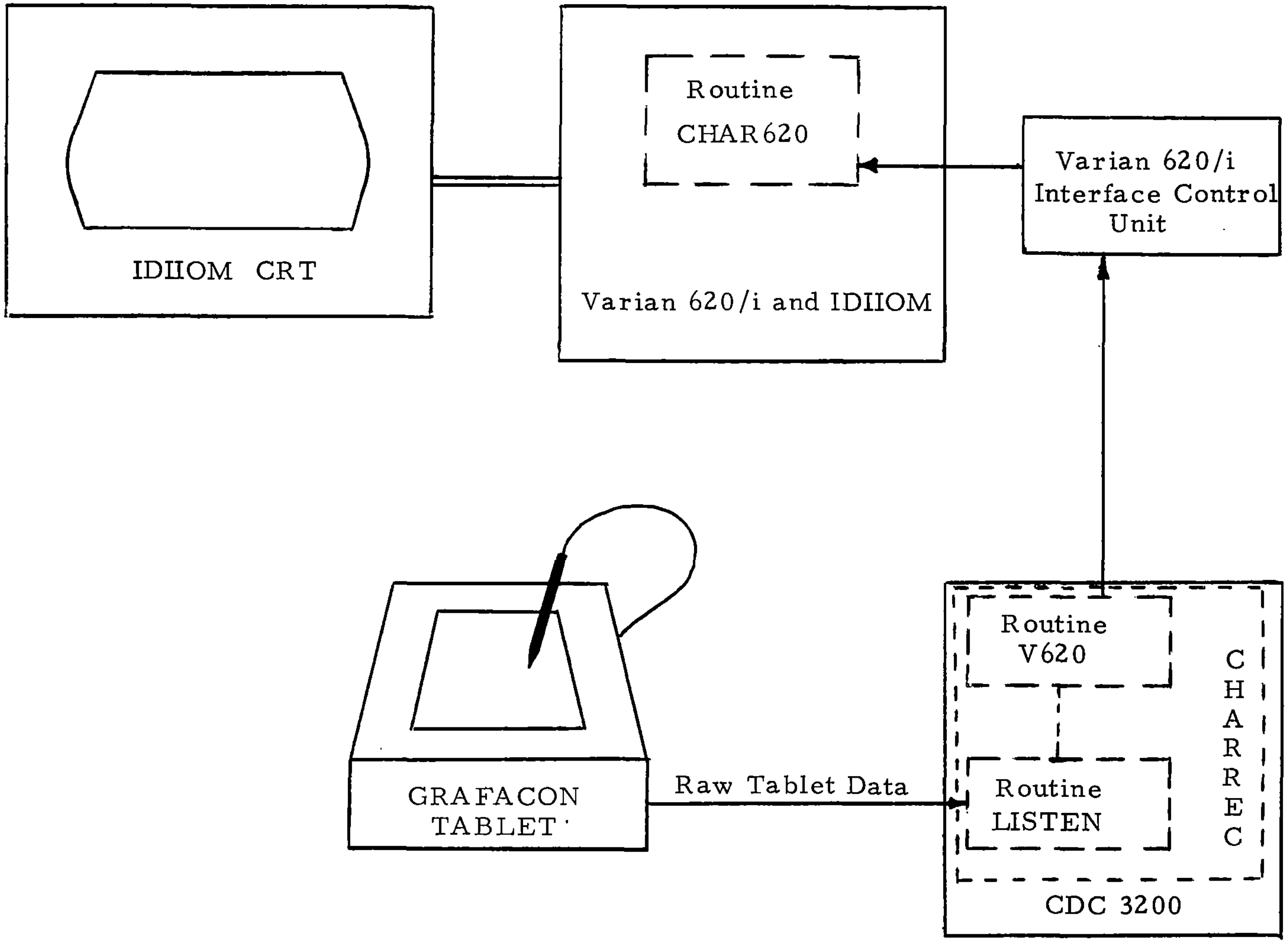


FIGURE A-1

GSFC Equipment Configuration

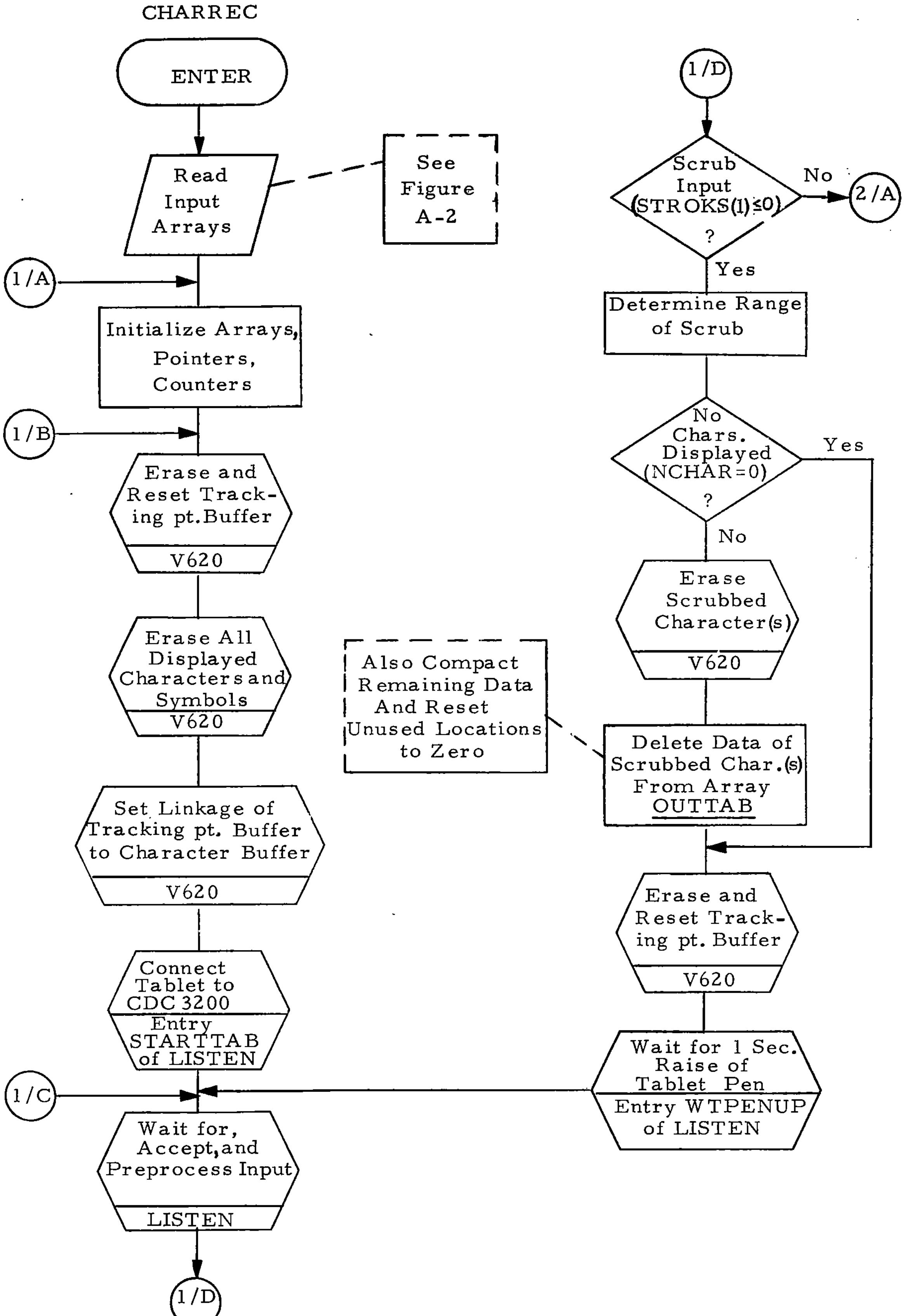

CHARREC
ENTER
Read Input Arrays
See Figure A-2
1/A
Initialize Arrays, Pointers, Counters
1/B
Erase and Reset Tracking pt.Buffer
V620
Erase All Displayed Characters and Symbols
V620
Set Linkage of Tracking pt. Buffer to Character Buffer
V620
Connect Tablet to CDC 3200
Entry STARTTAB of LISTEN
1/C
Wait for, Accept, and Preprocess Input
LISTEN
1/D
1/D
Scrub Input (STROKS(1)≤0) ?
No
2/A
Yes
Determine Range of Scrub
No Chars. Displayed (NCHAR=0) ?
Yes
No
Erase Scrubbed Character(s)
V620
Also Compact Remaining Data And Reset Unused Locations to Zero
Delete Data of Scrubbed Char.(s) From Array OUTTAB
Erase and Reset Tracking pt. Buffer
V620
Wait for 1 Sec. Raise of Tablet Pen
Entry WTPENUP of LISTEN

CHART A-1

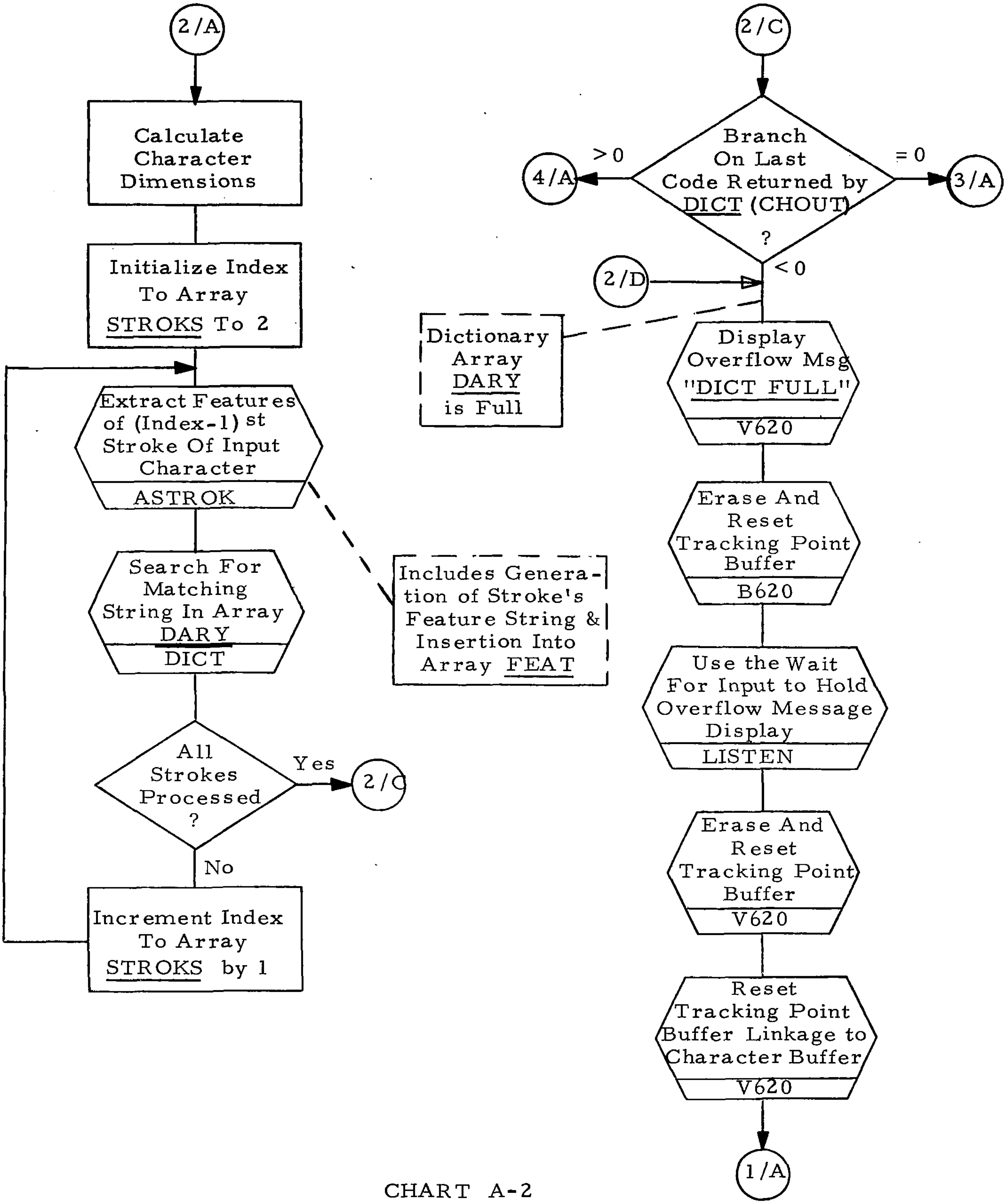

2/A
Calculate Character Dimensions
Initialize Index To Array STROKS To 2
Extract Features of (Index-1) st Stroke Of Input Character
ASTROK
Includes Generation of Stroke's Feature String & Insertion Into Array FEAT
Search For Matching String In Array DARY
DICT
All Strokes Processed ?
Yes
2/C
No
Increment Index To Array STROKS by 1
2/C
Branch On Last Code Returned by DICT (CHOUT) ?
> 0
4/A
= 0
3/A
< 0
2/D
Dictionary Array DARY is Full
Display Overflow Msg "DICT FULL"
V620
Erase And Reset Tracking Point Buffer
B620
Use the Wait For Input to Hold Overflow Message Display
LISTEN
Erase And Reset Tracking Point Buffer
V620
Reset Tracking Point Buffer Linkage to Character Buffer
V620
1/A

CHART A-2

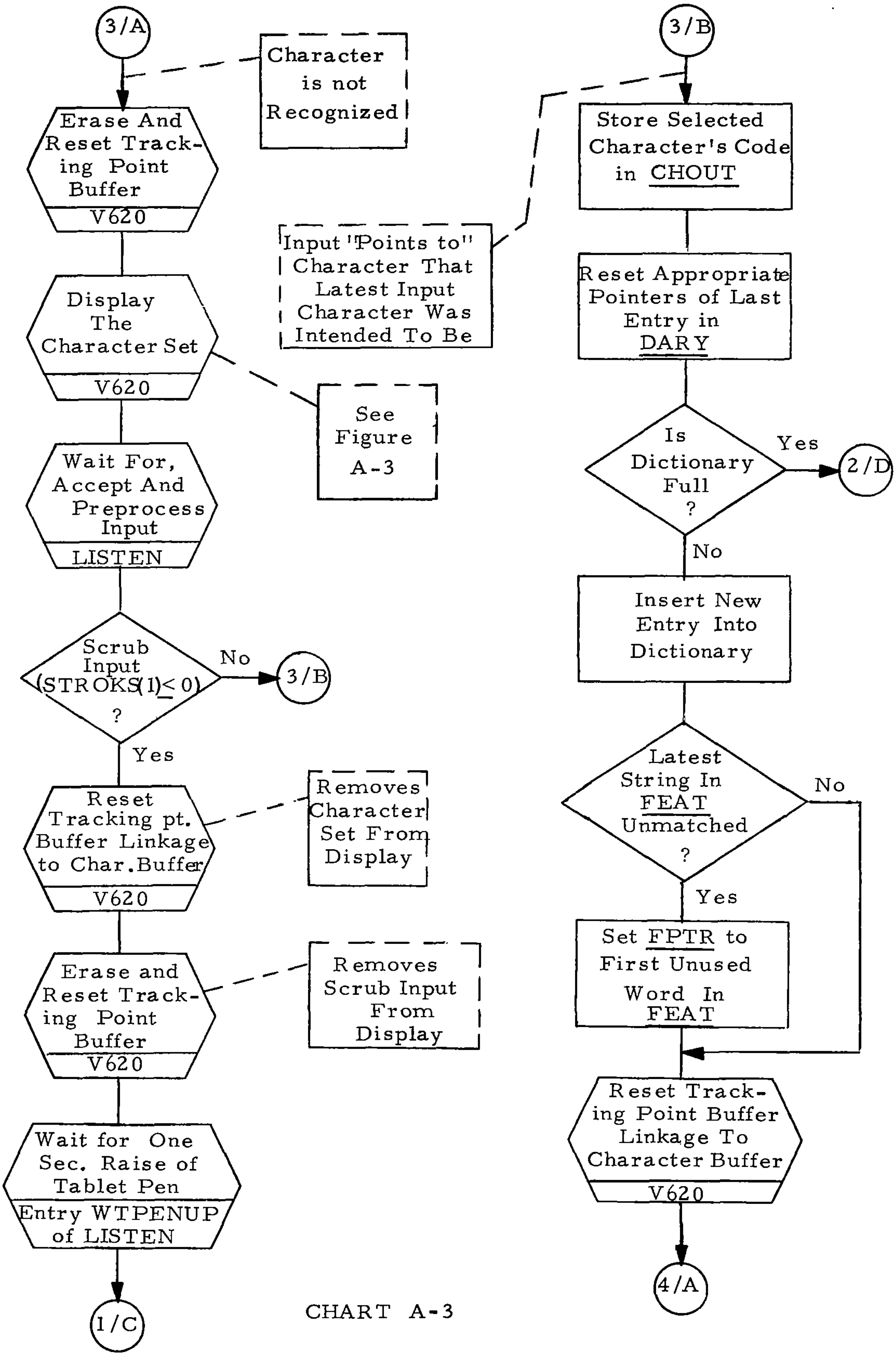

3/A
Character is not Recognized
Erase And Reset Tracking Point Buffer
V620
Display The Character Set
V620
See Figure A-3
Wait For, Accept And Preprocess Input
LISTEN
Scrub Input (STROKS(1)≤ 0) ?
No
3/B
Yes
Reset Tracking pt. Buffer Linkage to Char.Buffer
V620
Removes Character Set From Display
Erase and Reset Tracking Point Buffer
V620
Removes Scrub Input From Display
Wait for One Sec. Raise of Tablet Pen
Entry WTPENUP of LISTEN
1/C
3/B
Input "Points to" Character That Latest Input Character Was Intended To Be
Store Selected Character's Code in CHOUT
Reset Appropriate Pointers of Last Entry in DARY
Is Dictionary Full ?
Yes
2/D
No
Insert New Entry Into Dictionary
Latest String In FEAT Unmatched ?
No
Yes
Set FPTR to First Unused Word In FEAT
Reset Tracking Point Buffer Linkage To Character Buffer
V620
4/A

CHART A-3

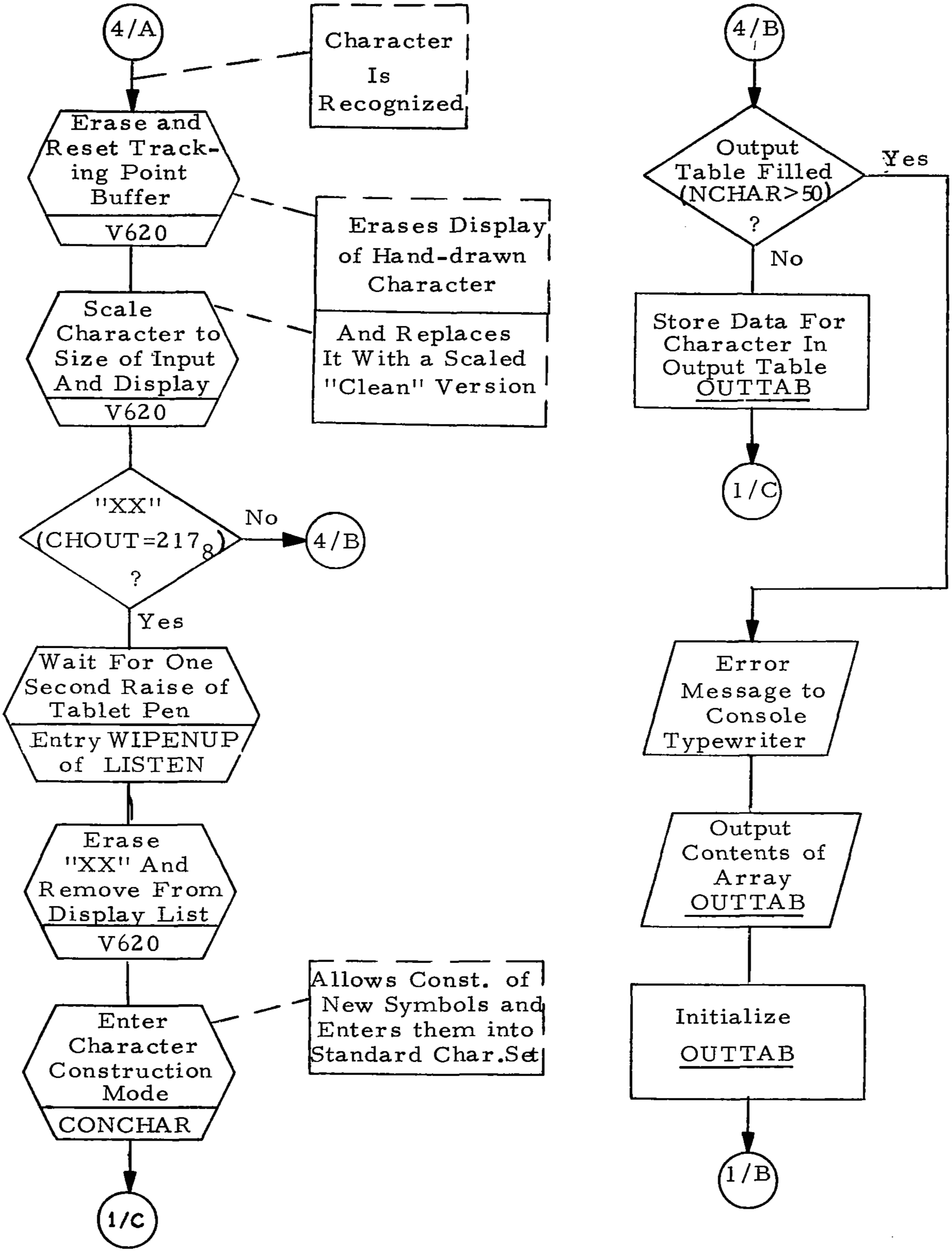

4/A
Character Is Recognized
Erase and Reset Tracking Point Buffer
V620
Erases Display of Hand-drawn Character
Scale Character to Size of Input And Display
V620
And Replaces It With a Scaled "Clean" Version
"XX" (CHOUT=217$_8$) ?
No
4/B
Yes
Wait For One Second Raise of Tablet Pen
Entry WIPENUP of LISTEN
Erase "XX" And Remove From Display List
V620
Enter Character Construction Mode
CONCHAR
Allows Const. of New Symbols and Enters them into Standard Char.Set
1/C
4/B
Output Table Filled (NCHAR>50) ?
Yes
No
Store Data For Character In Output Table OUTTAB
1/C
Error Message to Console Typewriter
Output Contents of Array OUTTAB
Initialize OUTTAB
1/B

CHART A-4

Curvature Threshold Array F(I)

I	1	2	3	4	5	6
F(I)	24	12	6	-7	-13	-25

For curvature >F(I), the curvature code = I

Orientation Code Array K (I, J)

J \ I	1	2	3	4
1	0	4	6	5
2	8	0	7	6
3	2	3	0	4
4	1	2	8	0
5	8	4	8	4
6	2	2	6	6
7	1	3	7	5

Heading array H (I, J) and ratio (slope) array R(I)

J	H (I, J)				R (I)
I	1	2	3	4	
1	0	16	0	-16	10.175
2	1	15	-1	-15	3.287
3	2	14	-2	-14	1.871
4	3	13	-3	-13	1.219
5	4	12	-4	-12	.821
6	5	11	-5	-11	.534
7	6	10	-6	-10	.303
8	7	9	-7	-9	.098
9	8	8	-8	-8	.000
10	8	4	8	4	2.414
11	1	3	7	5	.414
12	2	2	6	6	.000

FIGURE A-2

Input Data Arrays

A	B	C	D	E	F	G	H	I	J	K	L	M
301	302	303	304	305	306	307	310	311	312	313	314	315
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
316	317	320	321	322	323	324	325	326	327	330	331	332
0	1	2	3	4	5	6	7	8	9	+	[	]
260	261	262	263	264	265	266	267	270	271	253	333	335
Σ	π	√	∫	XX	~~V~~	=	<	>				
201	202	203	204	217	220	275	274	276				

The number below each character is it's octal code.

FIGURE A-3

The Standard Character Set

NASA-Langley, 1970 —— 8 CR-1634