AD

TECHNICAL REPORT ECOM-01901-27

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

FIFTH QUARTERLY REPORT

By

R. O. Duda *J. H. Munson*

June 1967

ECOM

UNITED STATES ARMY ELECTRONICS COMMAND · FORT MONMOUTH, N.J. 07703

CONTRACT DA 28-043 AMC-01901(E)

STANFORD RESEARCH INSTITUTE

Menlo Park, California 94025

TECHNICAL REPORT ECOM-01901-27 JUNE 1967

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

REPORT NO. 27, FIFTH QUARTERLY REPORT

1 FEBRUARY 1967 TO 31 MAY 1967

SRI Project 5864

CONTRACT NO. DA 28-043 AMC-01901(E)

Continuation of Contract No. DA 36-039 AMC-03247(E)

Prepared By

R. O. DUDA J. H. MUNSON

STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA 94025

For

U.S. ARMY ELECTRONICS COMMAND, FORT MONMOUTH, N. J. 07703

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The citation of trade names and names of manufacturers in this report is not to be construed as official Government indorsement or approval of commercial products or services referenced herein.

Disposition

Destroy this report when it is no longer needed. Do not return it to the originator.

ABSTRACT

This report describes the continuing development of preprocessing, character-classification, and context-analysis techniques for hand-printed text, such as computer coding sheets in the FORTRAN language.

We discuss preprocessing operations developed to find topological features such as the perimeter, the convex hull, concavities, enclosures, and spurs of a black/white quantized figure. We describe the present status of a program, TOPO 2, which classifies characters on the basis of these and other topological features. Finally, we detail the component functions of the LISP program developed to perform FORTRAN syntax analysis on a statement-by-statement basis.

CONTENTS

ILLUSTRATIONS

TABLES

I INTRODUCTION

This report describes the continuing development of preprocessing, character-classification, and context-analysis techniques for hand-printed text. The particular subject matter of our investigation is hand-printed FORTRAN text on standard computer coding sheets, with a 46-character alphabet. The reader is referred to the previous reports of this project for background and supplementary material.

In Sec. II of this report we resume the discussion of topological preprocessing operations that was begun in the Fourth Quarterly Report. We discuss procedures for finding the perimeter of a black/white quantized figure, its convex hull, its concavities, its enclosures and its spurs.

Section III describes the current status of the TOPO 2 program, which embodies the above-mentioned preprocessing operations and performs subsequent character classification through hand-coded evaluation functions. We also discuss the forthcoming use of the topological features from TOPO 2 as input to a learning machine.

The context-analysis phase of our effort is currently concentrated on analysis of individual statements, using the syntax of the FORTRAN language. In Sec. IV, we enumerate and briefly describe the action of numerous functions contained in the LISP program being developed for the syntax analysis.

II TOPOLOGICAL PREPROCESSING OPERATIONS FOR RECOGNITION OF HAND-PRINTED CHARACTERS

A. Introduction

We resume here the discussion of topological preprocessing operations that was begun in the Fourth Quarterly Report. In that report, we introduced a number of operations to be performed on an image in a quantized visual field. We also described programs written for the SDS 910 computer to perform these operations on a 24 × 24 bit, black/white quantized field:

Operation	Programs
Finding extent of figure(s) in field	EXTENT
Finding connected subfigures	CONN4, CONN8
Growing and shrinking figures	GROW, SHRINK
Dissecting a figure into connected subfigures	DISE48
Boolean operations performed in parallel on the elements of visual fields	ANDFIG, ORFIG, XORFIG, DIFFIG, CMPFIG
Translation of a field	RSHFIG, LSHFIG, USHFIG, DSHFIG

(A recent addition to this list is the program PIVOT, which pivots a 24 × 24 field about its main diagonal.)

The following terminology is used in this discussion: An object in the visual field is four-connected if any point in the object can be reached from any other point in the object by a path of unit steps in the four major directions (up, down, left, right) lying entirely within the object. An object is eight-connected under a similar definition, but with the path allowed to include steps in the four diagonal directions as well. J is the vertical coordinate in the visual field, running from 1 at the top to 24 at the bottom; K runs from 1 at the left to 24 at the right.

B. Contour Following (The Perimeter and the Shell)

A well-known result of recreational mathematics states that if one traces a finite two-dimensional maze, keeping the wall always at the right (or left) hand, he will eventually return to his starting point. If the starting point is on the outside of a connected figure, the contour that is traced is the outer boundary of the figure.

The contour-following operation, like that of finding connected subfigures, is a fundamental preprocessing operation and forms part of the basis for other operations to be described subsequently. A hardware implementation of the contour-following process, using a cycloidally looping flying spot to trace the character contour, has been used successfully at IBM Corporation in a device to read hand-printed numerals.[1]

When dealing with quantized images on a raster of points, the contour follower takes the form of a routine that steps from point to point along the edge of the figure. The natural output of the routine is thus an ordered list of coordinate pairs, rather than an image. The list may be condensed into an image containing those points in the list, but for some uses the ordering contained in the list is important.

We choose to begin the contour-following operation at a particular location on the figure, namely, the leftmost point of the figure on the 24 × 24 raster (and, in the case of ties, the bottommost of these points). The contour is followed in a clockwise direction. This fixed procedure assures us of finding a starting point on the boundary of the figure, and also makes possible some shortcuts in later processing. If the figure is not connected, only that connected component containing the starting point will be traced.

In a quantized image, the (imaginary) boundary wall may be thought of as running between the outermost figure points and the adjacent ground points. There are two ways, therefore, to trace the boundary

---

[1] E. C. Greanias, P. F. Meagher, R. J. Norman, and I. Essinger, "The Recognition of Handwritten Numerals by Contour Analysis," IBM Journal of Research and Development, Vol. 7, No. 1, pp. 14-21 (January 1963).

wall: on the inside, along the figure points, or on the outside, along the ground points. Let the perimeter be the list of figure points found by contour following along the inside of the wall, and let the shell be the list of ground points found by contour following along the outside of the wall. (We also use the names "shell" and "perimeter" to refer to the corresponding images composed of the points in the lists.) The starting point for the shell is the ground point just to the left of the (bottommost) leftmost figure point, where the perimeter starts.

We have so far only informally described the perimeter and shell. A formal definition would be somewhat complex and of mainly academic interest. We will give instead an operational definition corresponding to a computer routine for finding the perimeter.

1. Finding the Perimeter of a Figure

Step 1. At each stage there is a "current point" on the perimeter and a "current direction." Begin by taking the bottommost of the leftmost figure points as the current point. Take "up" as the current direction.

Step 2. Examine the eight neighbors of the current point, in the following order: clockwise, beginning with the diagonal neighbor just counterclockwise of the current direction. There are four possibilities:

	Current direction			
	Up	Right	Down	Left
Order of examination	1 2 3 8 X 4 7 6 5	7 8 1 6 X 2 5 4 3	5 6 7 4 X 8 3 2 1	3 4 5 2 X 6 1 8 7

The first figure point encountered is the next point on the perimeter, the "new"

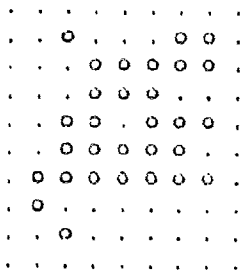
point. (If all eight neighbors are ground points, the figure consists of a single isolated point.)

Step 3. Take the direction of the step from the current point to the new point as the "new" direction. However, if the step was diagonal, take the major direction (up, right, down, or left) just counter-clockwise of the step direction, as the new direction.

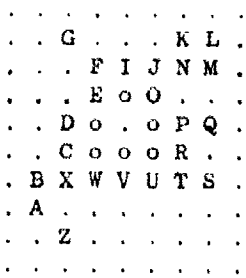
Step 4. Test for completion: if the current point and the new point are identical, respectively, to the starting point and the second point, the perimeter is complete. If not, replace the current point and direction with the new point and direction, and go back to Step 2.

The perimeter-finding process can be illustrated by applying it to the quantized figure shown in Fig. 1(a). The bottommost of the left-most points, shown as A in Fig. 1(b), is the original "current point." Since the starting direction is up, the first neighbor of A to be examined is a ground point. The second neighbor, B, is a figure point and is taken as the "new" point. The direction from A to B (namely, up) is the new direction. As the process continues, the perimeter points are found as shown, in alphabetical order.

Certain points on the figure may appear in the perimeter more than once. Thus point H (not shown) is the same as point F, and point Y is the same as point A. These duplicated points are not removed from the perimeter, for to do so would destroy the continuity of the perimeter path. This example shows why the test for completion must consider more than a single point; the perimeter returns to point A (i.e., point Y) prematurely, but it is not complete until the current point is A and the new point is B.



(a) Quantized Figure



(b) Figure with Perimeter Points

FIG. 1 FINDING THE PERIMETER OF A FIGURE

It may be noted that the perimeter cuts across an interior corner when possible (in the example, it goes from N to O instead of from N to J to O) but does not cut off an exterior corner (does not go from K to M).

We assert without proof that the list of figure points found by the procedure just described has the following properties, which correspond to the intuitive notion of "perimeter:"

Property P-1. Successive points on the list are adjacent (horizontally, vertically, or diagonally).

Property P-2. Each point on the list is four-adjacent (i.e., adjacent either vertically or horizontally) to an exterior ground point.

Property P-3. All figure points four-adjacent to the exterior ground are on the list.

Property P-4. The list goes "once around the figure, clockwise." In other words, the cumulative change in direction of the steps from one point to the next is 360° in the clockwise sense.

2. Finding the Shell of a Figure

A procedure, analogous to that just described for the perimeter, has been implemented for finding the shell of a figure. The shell begins at the ground point just to the left of the starting perimeter point and steps around the figure, in a clockwise direction, hugging the perimeter. An obvious duality exists between the perimeter of a figure and its shell; they are largely redundant, and only one need be found during pattern preprocessing. At present, we have discarded the shell in favor of the perimeter for several reasons. The perimeter contains fewer points, thus shortening the running time of subsequent computations. If a figure extends to the edge of the quantized field, the shell extends beyond the field, leading to annoying programming complications. Finally, the perimeter provides better starting information for the spur-finding procedure.

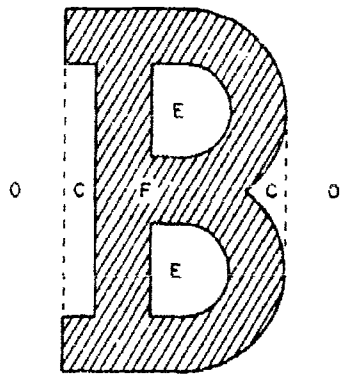
C. The Convex Hull

The convex hull H of a set F of points is the set of all points that are linear interpolations among points in F; that is, H is the set of points x such that

$$x = \sum_i a_i x_i$$

where the a are non-negative and sum to 1 and the x are in F. For a figure F in a plane, H may be characterized intuitively as the region enclosed by a rubber band stretched around F.

The convex hull of a character is of interest in preprocessing because it includes the figure and two types of ground regions: concavities, which are ground regions connected to the ground area outside the convex hull, and enclosures, which are isolated from the outside ground. Figure 2 shows a character image (F), together with its concavities (C) and enclosures (E). The convex hull is the union of the three regions. The concavities and enclosures (sometimes called "bays" and "lakes") are important features of a character. Thus, the convex hull defines what might be called the region of interest of the



CONVEX HULL H = F+E+C

FIG. 2 REGIONS ASSOCIATED WITH A CHARACTER IMAGE

character, and the boundary of the convex hull serves to separate the concavities from the outside ground (O).

Various computational methods could be used to find the convex hull of a quantized figure. Every ground point could be tested, for example, to determine whether it satisfies the defining condition with respect to the set of figure points. Certain shortcuts, such as the elimination of all but the extreme figure points, would clearly be needed to make this approach economical.

The method we have presently adopted makes use of the perimeter of the figure as its basis. Using the perimeter, we find the boundary of the convex hull as a list of points. This boundary forms a connected ring around the remainder of the convex hull, which can then be separated from the outside ground by a connectivity operation.

The value of the perimeter as a basis for finding the boundary of the convex hull is a consequence of two properties of the perimeter with respect to the extreme points of the character figure. The extreme points of a figure are those that lie on the boundary of its convex hull (The extreme points can also be defined as those that take on an extreme value, over the set of figure points, of any linear combination of the

spatial coordinates J and K.) For example, in the quantized, hand-printed letter B in Fig. 3 (Serial No. 200 in our data base), all eleven points in the bottommost row are among the extreme points because they all take on the maximum value of the vertical coordinate, J.

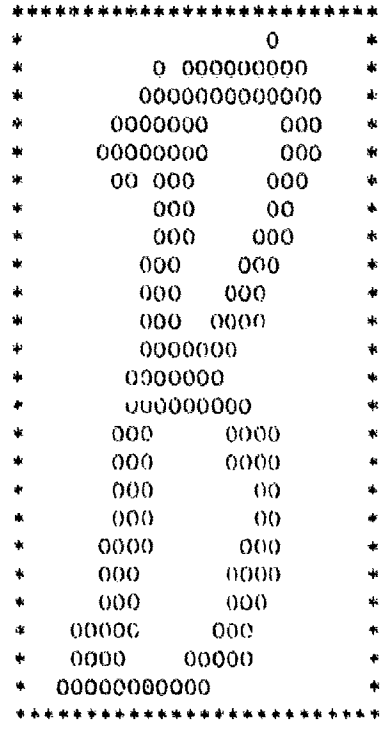


FIG. 3 A QUANTIZED, HAND-PRINTED LETTER "B"

Some extreme points are non-essential; they can be removed from the figure without changing the convex hull. This is true, for example, of all the points in the bottom row of Fig. 3 except the endmost two. Points such as these two, called corner points, are the ones that determine the convex hull of the figure. All other figure points can be discarded without changing the convex hull. (Corner points can also be defined as those figure points assuming unique extreme values of any linear combination of coordinates, i.e., extreme values not matched by any other figure points.)

The perimeter of the figure has the following two properties:

Property P-5. All of the extreme points of the figure (and hence, all of the corner points) lie on the perimeter.

Property P-6. At any corner point, the perimeter turns to the right.

A simple test tells whether the perimeter turns to the right at the Nth point. One calculates the unnormalized sine of the turning angle at the Nth point,

$$(J_{n+1} - J_n) \cdot (K_n - K_{n-1}) - (K_{n+1} - K_n) \cdot (J_n - J_{n-1}) \quad .$$

If this quantity is positive, the perimeter has turned to the right; if negative, to the left. If it is zero, the unnormalized cosine,

$$(J_{n+1} - J_n) \cdot (J_n - J_{n-1}) + (K_{n+1} - K_n) \cdot (K_n - K_{n-1}) \quad ,$$

is tested to see whether the perimeter has gone straight ahead or whether it has turned 180° to the right, as occurs at points G and Z in Fig. 1. (Note that no trigonometric functions need actually be evaluated.)

An efficient procedure for finding the boundary of the convex hull is therefore as follows: go through the perimeter list and mark all those points at which the perimeter turns to the right. The marked points include all corner points of the figure. (The marked points are denoted by X's in Fig. 4.) Begin with the first point on the perimeter, which is a corner point because it is the bottommost of the leftmost figure points. Go through the marked points, considering the vector from the starting point to each one. The marked point associated with the most nearly vertical vector is the next corner point, which is shown as B in Fig. 5. Using the vector from A to B as the test direction now (instead of the vertical), go through the remaining marked points and find the one, C, such that the vector $\underline{BC}$ makes the least angle with the vector $\underline{AB}$. C is the new corner point and $\underline{BC}$ is the new

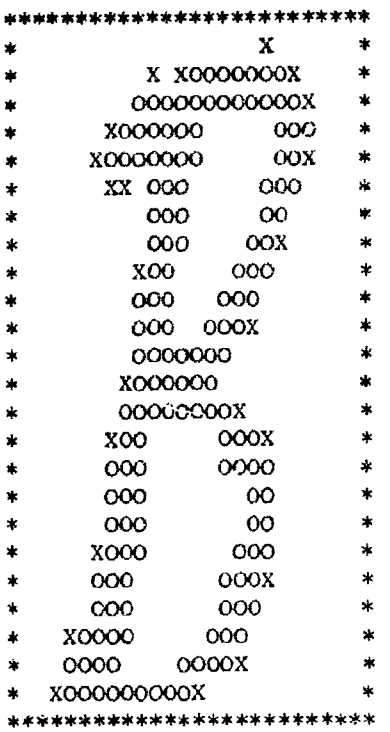


FIG. 4 HAND-PRINTED LETTER "B" WITH RIGHT-TURN POINTS MARKED

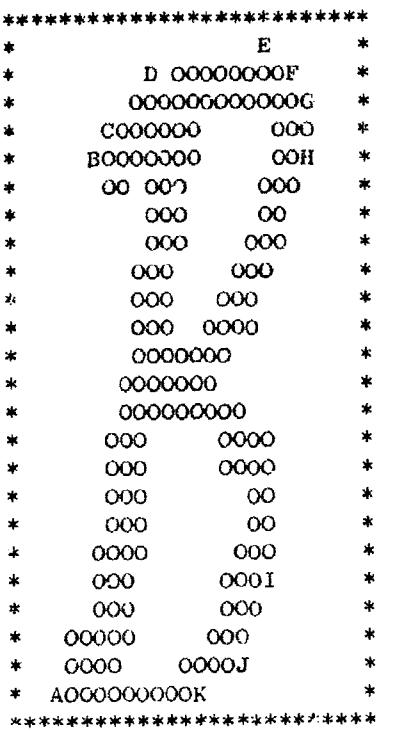


FIG. 5 HAND-PRINTED LETTER "B" WITH CORNER POINTS

test direction. Continue in this manner, finding corner points D, E, F, G, H, I, J, K, and finally A, which completes the ring of corner points.

It remains to fill in the stretches of the convex hull boundary between the corner points. Since a straight line between two points on the quantized grid does not in general pass exactly through grid locations, some latitude exists in choosing the paths connecting corner points. We have adopted the rule of building a path from one corner point to the next by taking horizontal, vertical, or diagonal steps as directly as possible in the direction of the target corner point but never outside of the line between the two points. Thus, in the example, the first step from A toward B must be diagonal, up and to the right, because a vertical step from A would go outside of the line AB. The

reason for this rule is that, by biasing the convex hull boundary to the inside, we greatly reduce the number of insignificant concavities found between the boundary and the figure as a result of irregularities in the perimeter. The ground point just to the right of corner point D is an example.

The completed convex hull boundary is shown in Fig. 6. It may be noted that this boundary consists partly of figure points (capital letters) and partly of ground points (small letters).

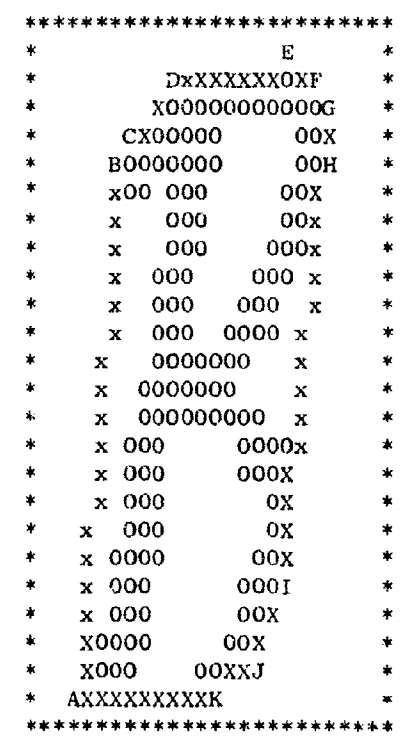


FIG. 6 HAND-PRINTED LETTER "B" WITH CONVEX HULL BOUNDARY

D. Concavities and Enclosures

Once the boundary of the convex hull has been established and transformed to an image on a 24 x 24 field, the concavities and enclosures may be found quite directly using the Boolean and connectivity

operations listed earlier. The convex hull boundary serves as a barrier (see Fig. 6) dividing the ground into two parts: the outside; and the concavities and enclosures (either of which may consist of zero, one, or more regions). The figure similarly serves as a barrier separating the enclosures from the rest of the ground.

We define the border of the field as consisting of those ground points lying in the outermost rows and columns of the field (J = 1 and 24; K = 1 and 24). That portion of the ground connected to the border is found by applying the four-connectivity operation; the remaining ground comprises the enclosures.

A separate image is formed consisting of the ground, minus the convex hull boundary. The portion of this image not connected to the border is found; this portion includes the concavities and enclosures. Subtracting the enclosures, which have already been found, leaves the concavities.

After the enclosures have been found as a group, they are separated by the dissecting routine DISE48. They are then ordered by size according to the number of points in each, found by the point-counting routine NUMPTS. The count of points in each enclosure is stored for future use, along with its top, bottom, left, and right boundaries, found by routine EXTENT.

The concavities undergo the same processing as the enclosures. In addition, a measure is generated of the magnitude of each concavity and the direction in which it faces out from the figure. These measures are analyzed and tabulated according to the eight major and secondary directions of the compass. This tabulation allows a subsequent decision routine to determine immediately if there is a concavity facing, say, up and to the right, without having to scan through all concavities.

E. Spurs

The spurs of a character are those strokes that end in an isolated tip. The letter X has four spurs, the letter O, none, and the letter S, one spur with the special property of having a tip at each end. The list

of perimeter points is used to find the spurs. (This description will largely duplicate one contained in the Third Quarterly Report, which described the finding of spurs from the shell of the figure. The full description is presented here for completeness, and to incorporate recent changes.)

Consider two pointers moving together down the list of perimeter points, with one pointer ahead of the other by, say, 14 places. As the pointers move, we calculate the Euclidean distance between the two perimeter points indicated by the pointers. Most of the time, this distance will be approximately 14 units. A sudden decrease of the distance between the two points to a minimum that is less than half its typical value indicates that the perimeter has gone around a sharp bend--i.e., has gone around the tip of a spur. The position of the spur tip, indicated by the perimeter point halfway on the list between the two minimum-separation points, is the primary spur feature used by the present classification routines.

Once the spur has been found, it can be traced by the "caliper method." Imagine that the legs of a pair of calipers are placed at the two minimum-separation points. The calipers are then "slid" along the spur by stepping the legs of the calipers along the perimeter, away from the tip. The calipers are moved as far as they can go without having to be spread by more than, say, seven units. In some cases, such as the numeral "6," the calipers will be obstructed by the body of the figure and must step. In other cases, such as the letter "S," the legs of the calipers will travel all the way along the figure and meet at the far end, indicating a "single-stroke" figure. The mid-point of the moving calipers traces out the backbone of the spur, and a list of the mid-point positions can be stored to represent the spur.

Considerable experience with the spur-finding procedure to date has shown it to be an effective method of finding spurs and locating their tips. However, as the method is presently constituted and used, there is all-or-nothing aspect to it as a consequence of the fixed criteria employed. As a result, there is a discontinuity in the

functional relationship of the spur features to the shape of the character image, as the spur test reverses its decision. It is impossible, however, to devise an all-or-nothing test that will accept all spurs (no matter how thick or garbled) and reject all acute corners and narrow loops. A spur test with the quality of "graceful degradation" is needed--one that produces a nonbinary "measure of spurness." This quality has been built into the measures of other features. The concavities, for example, are graded according to magnitude and degree of orientation in eight directions. Consideration is presently being given to variations of the spur-finding method that will yield the quality of graceful degradation.

III STATUS OF THE TOPOLOGICAL PREPROCESSING AND CLASSIFICATION PROGRAM (TOPO 2)

The topological preprocessing and classification program (TOPO 2), which was introduced briefly in the Fourth Quarterly Report, has been fully implemented on the SDS 910 computer. TOPO 2 is undergoing continuing testing and modification, so the picture presented here will be a "snapshot" of the status of TOPO 2 at the end of the period covered by this report.

A. Preprocessing

TOPO 2 takes in a 24 X 24 quantized character image from paper tape or magnetic tape and subjects it to the preprocessing operations described in this and the preceding report.

Since the main sequence of preprocessing operations is based on the presence of a single, connected character image, a special test is first made to detect the only character in our alphabet that should appear in two parts, the equal sign. If a character has two connected regions which fall within the allowed bounds of size and relative position, it is taken to be an equal sign and is not preprocessed further. Other characters, notably those with crossbars, also appear with multiple connected regions at times. Therefore, any multiple-region character failing the criteria for an equal sign is forcibly reduced to a single region, first by bridging the gap between regions, or, if the gap is too great, by deleting the lesser region(s). Conversely, an equal sign may appear as a single connected figure, in which case it passes through the usual processing, and may be evaluated as an equal sign by appropriate tests.

The preprocessing operations determine the following for the connected figure: its vertical and horizontal extent; its slant and curvature if it is a tall, narrow figure; its concavities, its enclosures, its spurs, and its profiles as seen from the four major directions. In the course of these operations, the perimeter and the convex hull are also found.

B. Intermediate Steps

The preprocessed information goes through two intermediate steps before it is ready for the classification routines. The purpose of these steps is to predigest the information for the classification routines. The need for such predigestion may be seen by considering that 46 classification routines are required, and each routine will consider an average of perhaps a dozen items from the preprocessed data. The writing of half a thousand pieces of computer code would be extremely burdensome, so it is vital that the preprocessed information be properly organized and appear in the form of variable quantities that can be used directly by the classification routines.

The first intermediate step consists of rearrangement and tabulation of some of the preprocessed information. The enclosures are sorted by size, so that a classification routine can easily reference the largest (hence, the most definite) ones. A tabulation of the concavities is made according to their magnitudes and the directions in which they face. A similar tabulation is made for the tips of the spurs, specifying the degree of closeness of the closest tip to the top, to the upper right corner, to the right side, etc.

In the second step, one or more confidence-loss functions (CLF's) are calculated for each of the important preprocessed quantities. Each CLF indicates the loss of confidence that the character in question is to suffer if a given attribute is expected of it. The loss function transforms the original measure of a preprocessed quantity into a numerical quantity usable by the classification routines without further computation.

As an example, we consider the CLF's pertaining to up-facing concavities. The concavities have been found by the preprocessing routine, and a measure of the presence and magnitude of an up-facing concavity has been tabulated. This measure, which we may call MCONC(UP), is the maximum (over all concavities) of the up-directed component of a vector indicating the size and direction of each concavity.

Figure 7 shows three CLF's based on MCONC(UP). Function LYCONA, shown in Fig. 7(a), has this interpretation: there is no loss in confidence if there is no up-facing concavity. If, however, there is indication of an up-facing concavity, a loss is incurred which increases

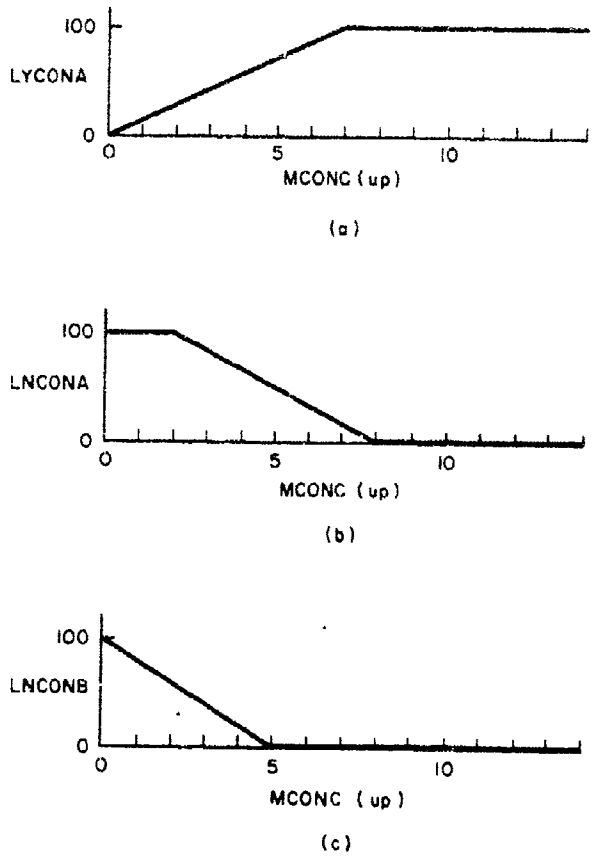


FIG. 7 CONFIDENCE-LOSS FUNCTIONS

with increasing up-strength. By the time MCONC(UP) reaches a value of 7, there is practically no doubt that the figure contains a true, significant up-facing concavity, and the full loss is incurred. Function LYCONA is thus appropriate for inclusion as a term in the evaluation functions of character categories such as the numerals 1, 2, 3, 5, etc. and the letters A-G, I, O, P, etc., where one never expects an up-facing concavity.

Figure 7(b) shows function LNCONA, which represents a loss in confidence if there is not an up-facing concavity. LNCONA is thus appropriate to the evaluation functions of categories such as the letters U and V. Function LNCONB, shown in Fig. 7(c), also represents a loss in confidence in the absence of a concavity, but more conservatively than does LNCONA. LNCONB might be appropriate to those characters with smaller (and thus more readily obscured) up-facing concavities, such as the open-style numeral 4 and letters H, M, N, W, X, and Y.

As the example shows, more than one CLF can be applied to a preprocessed feature, each CLF reflecting a commonly used evaluation of an attribute of the character. The ramp nature of the CLF's represents the quality of graceful degradation that we believe is vital for features used in character recognition, especially when the "recognition" of characters is really the assignment of numerical confidences to categories for input to a text recognizer.

C. Classification

The classifier section of the TOPO 2 program yields, not a unique category assignment for each character image, but a list of alternative category assignments, each with a confidence measure. Each confidence measure may be considered an estimate by the classifier of the confidence (or probability) that the observation (the character image) represents a member of the associated category. The confidences range from zero to 100; in practice, most of the 46 values generated for a character are zero. The present version of TOPO 2 calculates the confidences independently, and does not normalize their sum to 100 or any fixed number. Hence, the confidences are not true probability estimates.

The basic expression evaluated for each category is a simple sum of confidence-loss functions for those attributes pertinent to the category. For example, the total loss for the letter-B category may be the sum of losses incurred for: an up-facing concavity (hence, LYCONA), a down-facing concavity, no right-facing concavity, spur tips on the right side, lack of an enclosure, and lack of a second enclosure. The sum of losses (limited to 100) is subtracted from 100 to give the basic

confidence measure. Ideally, a well-printed B will satisfy all the requirements well enough to incur no losses and achieve a confidence of 100. Most other characters will incur losses totaling 100 or more and be given a confidence of zero in the B category.

If the variations in hand-printed material were adequately covered by a straightforward selection of topological features, the classification routines could be assembled with ease. (Furthermore, this or other methods would have solved the character-recognition problem long ago.) A look at Fig. 6, however, indicates some of the problems with the B category and gives just a hint of the gamut of problems encountered in hand-printed material. Although a B nominally has a straight left side, the example in Fig. 6 has two serifs that form a large concavity on the left. The serifs are almost large enough to be found as spurs, as they are in many B's. In others, the enclosures are broken open or merged into one large enclosure. Thus, contrary to naive expectation, no loss should be incurred for left-side concavities or spurs, and only a limited loss for the absence of two enclosures.

In a program such as the current version of TOPO 2, where the evaluation functions are designed by a human programmer, considerable experience with real data is required to improve the selection and weighting of the CLF's. The present effort to develop TOPO 2 is largely devoted to this task, through a repeated cycle of programming, running, observations, and evaluation. The human designer has recourse to several avenues of flexibility. He may ameliorate the effect of a CLF on a particular category by dividing it by some factor N, so that the maximum loss incurrable as a result of the corresponding feature is 100/N. He may turn to the preprocessed data to devise new, tailor-made CLF's.

Finally, the designer may turn to the original image or any of the information that has been generated during the processing, to perform any calculation required in a special case. In TOPO 2 this last recourse is used, to perform a detailed discrimination, only when the basic confidence measures have singled out a particular set of categories. For example, a detailed calculation to adjust the relative confidence

of the 5 and 8 categories is performed only when those categories receive non-zero confidence measures from the basic evaluation. There are presently about twenty of these detailed calculations.

D. Classification by a Learning Machine from Topological Features

Given the set of preprocessed topological features from TOPO 2, which we have suggested are highly suitable for classification, it is natural to ask whether these features can be input to a learning machine to perform the classification. If the learning machine takes the form of a piecewise linear machine, the dot product sums formed for each category can be related to the desired confidence measures. In any case, the effectiveness of the topological-feature plus learning-machine combination can be examined by the usual method of counting first-choice-correct classifications.

It seems best to present the feature information to the learning machine in the form of CLF's, since they are single variables, bounded and continuous, and they reflect our estimates of worthwhile parameterizations of the preprocessed information. The basic CLF's of TOPO 2 cover most of the preprocessed information, but to give the learning machine a better chance to make the special discriminations such as 5-8, we shall also give the machine the results of the special calculations when they are performed. This will put the revised TOPO 2 program that prepares the learning-machine features in the curious position of having to calculate the existing hand-designed basic confidence measures solely in order to know which special feature calculations to perform. It is reasonable to use this "wasteful" procedure for the purposes of initial experimentation, and to consider other techniques if the experimental results are encouraging.

A version of TOPO 2 which will prepare the required feature vectors is currently being written. A large number of characters will be processed through this program, and the resulting data used to form training and testing sets for the learning machine. An attempt will be made to use as learning-machine training data those characters that have been used for "human training" during the development of the evaluation functions in TOPO 2.

We shall report subsequently on the results of the learning-machine experiments, which will afford a two-way comparison: with the results of the TOPO 2 evaluation functions working on the same preprocessed data (see the next section), and with the results of the same learning machine working on the very different preprocessed data provided by the optical mask preprocessors, described in the preceding reports of this project.

E. Experimental Results

The TOPO 2 program was used, in the form that it had at the end of the period covered by this report, to process several sets of hand-printed data. Although the output of TOPO 2 consists of lists of confidence measures, the simplest and most familiar performance measure is obtained by specifically classifying the character in the category with the highest confidence. The resulting simple measure (classification error rate or success rate) may be used for comparison with other types of experiments.

1. Authors 1-20

This experiment was performed on a set of 920 characters, consisting of the first FORTRAN alphabet coded by each of the first twenty authors from whom we collected our data base. The error map for this experiment is shown in Table I. The error map shows a blank for each character correctly classified by TOPO 2, i.e., each character for which the correct category received the highest confidence score. For each character incorrectly classified, the entry indicates the character category receiving the highest score. Thus, the error map contains the same information as a confusion matrix, and also points out the individual misclassified characters.

The overall classification error rate is 187/920, or 20 percent. There is a wide variation in the error rates for individual authors--all the way from 2.46 to 19.46. Low error rates appear to be correlated with neatness of printing, stroke sharpness (high contrast on the page and small ratio of stroke width to character size), and lack of ornamentation.

Table I

CLASSIFICATION ERROR MAP FOR AUTHORS 1-20

	Author																				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Total
1					I							/								/	3
2			Z		I								Z	]						Ø	5
3								1	7												2
4									Y				+	P	$				U		5
5								6	S		I	S									4
6		G		G					1					G					F		5
7											T	2									2
8		B					Ø		P	9		Ø							B		6
9		4		F			D		S					P							5
Ø			B															Q			2
A				,												Ø				9	3
B		8						Ø	9	8				8	8				Ø	8	8
C		G				L			Ø				G						6		5
D								8			B							O			3
E						6						G				F					3
F				5																	1
G			6					6	Q				Ø	6		6				4	7
H								M		M		N						N		*	5
I		T			5			P							5			F	S	F	7
J				3										3	5			,	5		5
K		Y						4							X						3
L																				6	1
M							A		9					P				H			4
N					P								U	W							3
O												U								U	2
P								7	8					T		9		Y	7		6
Q		B							F				P	8		P					5
R								A	P						A					P	4
S				5				5									5		5	8	5
T																		5	5		2
U								Ø								V					2
V								1								Y		U			3
W									U				N							N	3
X									P	*				,	Y					4	5
Y																					0
Z																*					1
[						6						6	C			6		6	6		6
=										-					-						2
*	P	5	Y	X	P		Ø			P	F			X	-	Y	P				13
/		1	1																		2
+		*									*			F							6
-																					0
.																					0
,				7									J								2
$	*	8		8		8		8	Ø	8			*	Ø	J	,		Ø	Ø	T	14
]		2							3	3			,			,	,	J			7
Total	2	13	5	9	5	4	1	11	19	9	5	8	11	15	11	12	3	12	12	14	187

There is also considerable variation in the error rates for individual character categories. Some characters, notably those with the simpler shapes such as the period, minus sign, 7, and L, have error rates much lower than the average. The two most "cluttered" characters, the asterisk and the dollar sign, have high error rates because they are generally the most poorly printed. The asterisk often appears as a curiously shaped or formless blob. Of the twenty dollar signs, only one had the two enclosures that would normally be expected between the S-part and the vertical stroke. Devising features applicable to these categories is extremely difficult.

The twenty alphabets classified in this experiment may be considered training data, in that they contained the characters processed and studied by the human designer during the development of TOPO 2 to its present form.

2. Authors 21-40

The data set for this experiment consisted of 920 characters, the first alphabet coded by the 21st through 40th authors from whom we collected our data base. These twenty authors were largely personnel of the SRI computer center, whereas the first twenty authors were personnel of our own laboratory, the Artifical Intelligence Group.

The error map for this experiment is presented in Table II. The overall error rate is 266/920, or 29 percent. The alphabets used here represent independent test data. They were neither examined nor processed by TOPO 2 before this experiment.

A number of possible causes may have combined to produce the difference in error rates between this and the previous experiment. (The combined statistical uncertainty is on the order of 3 percent, which is insufficient to account for the difference.) First, of course, is the difference in performance between training and testing data, which we have usually encountered in our learning-machine experiments. Second is the possibility that more care in printing was taken by the personnel in the Artificial Intelligence Group, because of their greater proximity

Table II

CLASSIFICATION ERROR MAP FOR AUTHORS 21-40

	Author																				
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	Total
1					/				/						/	/		/	5		6
2		Z				I	R	R			?		Z		F	8		I	Ø	,	11
3	1	,		1					8				|	$	?		J		J		9
4	Ø							9	P						Ø			*			5
5	S				F	X					X			F	J				G		7
6	L	G				8		8	G							G		G	G		8
7																J		T		2	3
8		B	B		B				Ø		P		Ø	$	B			B		Ø	10
9	8	8	4	4				8			8	8	7				8	F	4	Q	12
Ø	$	P																			2
A	Ø														9	M		Ø		*	5
B				R			8	Q	Q				?			Q		8		P	8
C	6	G	G		G				6			L	L						G	8	9
D				O			8										O		Ø		4
E					F					=											2
F	6						T	K												E	4
G	6		Ø	6			6			X				C		F	6				8
H	P			M		M	N		N	N			K		N	4		N	N		11
I	2		2	J							F		I		J						6
J			8						3				|								3
K	R			X															*		3
L					G									E					P		3
M					N				*												2
N							M				P		U							P	4
O	D						P						D								3
P	8		7	?									8		7			7		Z	7
Q		4		9	4			O					O		P			P	P		8
R	Q						P		P						P			?			5
S	5		5	I	5	5	5	5	5	I				5	0	I	6	5		5	15
T					J															2	2
U							J		6			Y				K		K		P	6
V	J						Y		U								Y				4
W							J			N				H	N	4					5
X	1																				1
Y	4		P						2							J					4
Ø				4				$	L	*				*				P		I	7
|	L	1															6			6	4
,							-								-				-		3
*						X	8	0	P			P	X	X			Y				8
							I													I	2
+	8		8				7	*	I		*	*			*		*		*	4	11
-																					0
.										I								G		-	3
5	1								1									J	J		5
$	8	0	8	0		0	8	0	8	8	0				0	0	0	0	*		15
|	,						,												,		3
Total	25	10	11	13	11	7	19	12	20	8	8	5	14	9	17	13	10	19	17	18	266

Note: * indicates all 36 confidence scores zero.

to the project or the way in which the coding task was presented. We tried, of course, to minimize this difference through our instructions to the authors. Third, the document scanning, which required the assistance of a human operator, may have been performed less carefully as time went on. A fourth possibility, that of a general difference in coding styles, appears quite unlikely because of the similarity of backgrounds between the two groups of authors. Finally, there is the possibility of statistical fluctuation on an author-by-author basis. To the extent that each person prints with a characteristic level of neatness, clarity, etc., the characters printed by him are not statistically independent events. The governing statistics then are those of 20 independent authors, not 920 independent characters.

3. Authors 41-49

The data set for this experiment consisted of 414 characters, the first alphabet coded by the 41st through 49th authors. These nine authors were personnel of the computation staff at Fort Monmouth.

The error map for this experiment is presented in Table III. The overall error rate is 100/414, or 24 percent. The nine alphabets represent independent test data. It may be noted that the error rate on these data lies closer to that on the "training data" (Authors 1-20) than that on the test data of the previous experiment (Authors 21-40).

The next two experiments were performed on quantized hand-printed data supplied to us through the courtesy of Noel Herbst and his colleagues at the Thomas J. Watson Research Center of the International Business Machines Corporation. The data supplied were quantized with approximately 30 x 30 resolution. We brought the data down to our own raster size of 24 x 24 by casting out rows and columns wherever necessary, and reformatted the data records to match our existing programs.

4. Casey's Data: Letters and Numerals

A data base containing hand-printed letters and numerals was collected by Richard Casey from personnel of the IBM Research Center, in much the same way that we collected our data base. Each coder wrote

Table 111

CLASSIFICATION ERROR MAP FOR AUTHORS 41-49

	Author									
	41	42	43	44	45	46	47	48	49	Total
1	,									1
2						I	R		$	3
3			J			2				2
4								T		1
5				8						1
6			G					G		2
7										0
8		B	B			B				3
9	P				4	4			4	4
0				•						1
A				R						1
B							D	8		2
C	G			[	?	6	L			5
D	D								8	2
E	F				8					2
F					6				6	2
G				6	6			6		3
H	,			P	P				P	3
I		F	J				F			3
J			*			I				2
K			X							1
L			6				F	E		3
M										0
N	M	H		M		H				4
O				D	D					2
P				8	8				8	3
Q							4	6	8	3
R							P			1
S		5		5		5	8	X	/	4
T									8	1
U							4			1
V										0
W				M	0			N		3
X		R								1
Y										0
Z				]						1
[		I					(			2
	8			,				8		3
*	P	X	Y		P					4
							*			1
(				[						1
										0
.										0
,	J			I		I	J	I	J	6
$	0			6	0	0	0	I	0	7
1	4					,				2
Total	12	7	8	10	12	10	13	11	11	100

100 - 11 = 24 percent

the 26-letter English alphabet plus the ten digits. Since some of the characters were omitted in scanning, however, the data were not precisely organized by authors. Instead, the first twenty samples of each category on the data tape were used.

The TOPO 2 program was treated as a 36-category classifier for this experiment by ignoring all responses from the categories of the ten special FORTRAN-alphabet characters.

The data set for the experiment included 700 characters; the 36th category, letter Z, was not processed because the coders were not instructed to put a crossbar on Z. A listing of the errors is given in Table IV. The overall error rate was 150/697, or 21 percent. This rate does not include three numeral 1's printed with serifs, and it counts as correct 18 unslashed numeral Ø's recognized as O's and six unslashed I's recognized as numeral 1's.

If the results of the experiment on Authors 31-40 are re-evaluated, considering only the 36 alphabetic and numeric categories, the error rate is found to be 103/380, or 29 percent. The difference in rates is due to some combination of the factors listed: different coding environments, different scanning methods, possible difference in coding style, and possible statistical fluctuations.

The performance of TOPO 2 on Casey's data has eased our concern that the program might be "provincial," i.e., slanted toward our own data and methods. This experiment has demonstrated the ability of TOPO 2 to work on data generated by a different scanning system from different source documents, at a completely independent research organization.

5. The Backroom Data

The second data base provided by IBM was the "backroom data," consisting of hand-printed numerals produced in the course of routine operations by four inventory clerks in a department store. These data are considered quite "clean"; the clerks in the back room work in a more controlled environment and are probably subjected to a lesser variety of pressures than the sales staff. When the quantized numerals are displayed

Table IV

CLASSIFICATION ERROR LISTING FOR CASEY'S DATA

Category	Misclassifications	Total
1	- - -	0
2	3ØIØ5	5
3	P85PJ G	6
4	YUY1	4
5	FFF	3
6	GG8GG GLF	8
7	Y?2	3
8	BBBB6 ØØ	7
9	84Ø44 7	6
Ø	- - -	Ø
A	MP	2
B	QQDØD ?OO	8
C	G8G	3
D	OGQ	3
E	GG8GG 6F	7
F	GK	2
G	66U	3
H	MMMPM UPM	8
I	F8F	3
J	UVIXU OØ	7
K	LØG4	4
L	?FRX? EE	7
M	HN	2
N	MMPM	4
O	UØD	3
P	- - -	Ø
Q	WOOOP PP4BB	10
R	K	1
S	55555 5155P J555	14
T	Y	1
U	XKK	3
V	UU?	3
W	HNNUN	5
X	Y4Y4	4
Y	4	1
Z	not processed	- - -
		150, 697 21%

Note: ? indicates all 36 confidences were zero.

at the computer, they are clearly much better formed than the numerals in our own data base.

The data set used in this experiment consisted of 100 examples of each of the numerals 1-9. Numeral zero was not included, since it is not slashed in this data base. (We could, in the future, include the zeroes and treat them as letter O's.) A listing of the errors is given in Table V. The overall error rate was 33/900, or 3.7 percent. This error rate was achieved, it should be emphasized, by a classifier in which the design effort was spread over 46 categories instead of 10, and for which none of the backroom data base was used in the design process.

Table V

CLASSIFICATION ERROR LISTING FOR BACKROOM DATA

Category	Misclassifications	Total
1	- - -	0
2	- - -	0
3	52222	5
4	- - -	0
5	- - -	0
6	?8880888?88	11
7	22	2
8	?744?	5
9	4487445548	10
0	not processed	-
		33/900=3.7%

Note: ? indicates all 10 confidences were zero.

F. Plans for the Near Future

Our attention is presently turned toward the experiments in which a piecewise-linear learning machine will be used to classify the feature vectors obtained from the modified TOPO 2 program. The results of these experiments should provide valuable insights as to whether the pre-processing or the classification is the area in which our development

effort should next be concentrated. In addition, we already have knowledge of areas particularly requiring improvement. In preprocessing, we have discussed in this report the need for spur-finding features with the quality of graceful degradation. The classification error maps show groups of often-confused characters requiring further refinements in the classifier and new features from the preprocessing. There is a strong suggestion from the experimental results that the original confidence loss functions were applied too harshly in general. We are considering revamping the program to enhance the overall quality of graceful degradation by providing more conservative CLF's to a greater number of features.

Finally, we are beginning to explore the topic of adjusting, or tailoring, the classifier to fit the printing of the individual author. We have purposely held off on this approach, which will involve additional training effort or complexity. But there are instances (numeral 1's and slashes being the classic example) where different characters printed by different authors are similar or identical. Furthermore, in most cases the variations of a character produced by a single author are far less than those for the whole population. The learning-machine experiments will provide a convenient framework for exploring the tailoring approach. We will be able to "enrich" the training set with data produced by the test author(s) in any amount from 0 to 100 percent.

IV DESCRIPTION OF PROGRAMS FOR SYNTAX ANALYSIS

A. Introduction

Our general approach to the use of the syntax of FORTRAN to resolve ambiguities in character recognition was described in the Fourth Quarterly Report. Our work to date has been restricted to isolated statements; thus, no information from other parts of the program, such as variable names or statement labels, has been used in trying to resolve a particular statement. While much remains to be done to complete this effort, this work has progressed sufficiently to warrant documenting it in more detail.

The most accurate description of the program is given by the listing of the program itself, and this listing is included in the Appendix. The listing gives the exact definitions of all of the LISP functions used by the syntax analyzer, together with values of parameters. Such a listing can be traced without too much trouble by anyone familiar with LISP once the general structure of the program is known and the general purpose of the most important functions is known. The purpose of this section is to supply this needed background material.

B. Assumptions

Although our approach to using syntax to resolve ambiguities should be applicable to FORTRAN generally, the details depend on the particular dialect used. Our program is written specifically for FORTRAN II as described in the Scientific Data Systems reference manual SDS 900003C. Answers to all questions of what is or is not syntactically correct are based on the Backus-normal-form description of SDS FORTRAN II given in Appendix D of that manual.

The input to the program is assumed to be a list of lists of character-confidence pairs obtained from the classifier. Thus the input, called a P-list, has the form

$$P = (L_1\ L_2\ \ldots\ L_n) \quad ,$$

where the ith top-level element L_i corresponds to the ith character in the statement. Each top-level element is a so-called L-list and has the form

$$L = (\ (\ Char_1\ Con_1\ )\ (\ Char_2\ Con_2\ )\ \ldots\ (\ Char_m\ Con_m)\ )$$

where $Char_i$ is the ith choice for the category of the character, and Con_i is the confidence for that choice. For example, the scanning of an END-statement might result in the following P-list from the classifier:

```
P = ( ( ( E 60) (G 20) (6 10))
      ( ( N 50) (W 20) (H 10) (Y 10) )
      ( ( O 50) (D 40) ) )     .
```

When we are only interested in the alternatives for each character, it is simpler to use so-called A-lists having the form

$$A = (K_1\ K_2\ \ldots\ K_n) \quad ,$$

where each K-list has the form

$$K = (Char_1\ Char_2\ \ldots\ Char_m) \quad .$$

We have attempted to use this notation consistently, so that by looking at the arguments of the function one can tell the kind of data-structure it is meant to receive. For this same reason, we have tried to use S to denote a list of characters meant to be a meaningful part of a FORTRAN statement, and have used names beginning with T for thresholds. Exceptions exist, however, and the reader should consider this primarily as a general guide.

As mentioned previously, we assume that the P-list from the classifier corresponds to a single FORTRAN statement. This may correspond to one or more lines on a coding sheet, depending on whether or not the statement was continued. We assume that the minor task of checking for the presence of continuation characters in Column 6 and properly stringing the continued lines together to yield a P-list for a complete statement has been done by the time our current top-level function is applied. We also neglect the label field, assuming that the P-list begins with the character in Column 7.

Finally, there is the matter of spaces (blanks) in the field. In principle, all spaces could be deleted from statements other than FORMAT statements, since, if no errors were made in recognizing the characters, spaces would be deleted by the compiler anyway. However, spaces are very valuable in ambiguous situations. For example, the recognition of

```
DO 23 I=1,7
```

as a DO-statement is greatly aided by the presence of the spaces. Thus we have decided to retain spaces in the P-lists during the initial analysis of the statements, removing only repeated spaces. This leads to occasional annoyances, such as the need to treat GO TO and GOTO separately, or the need to treat IF (and IF(separately. However, this is a relatively small price to pay for the significant improvement in the ability to recognize statement types.

C. General Program Structure

1. Overall Structure

The present syntax analysis program is a function called st[p]. Its argument, p, is the P-list obtained from the classifier, and its value is either an S-list representing our resolution of the statement, or an error message indicating the inability of the program to resolve the statement. st can be thought of as a main program which starts the job of passing the program to various subroutines. Its

formal definition is given by*

$$st[p] = stype[\ space[p]\ ] \quad .$$

Here space merely removes leading and repeated internal spaces from the P-list. stype is the main function for determining the statement type. Its operation depends on the fact that all FORTRAN statements other than the arithmetic-assignment statement begin with a "control" word, such as DO, FORMAT, DIMENSION, etc. It uses a predicate gm[p ; m ; h] which measures how well the front of the P-list matches one of these words. To be more specific, gm is true if the average confidence of characters in p corresponding to characters in the control word m exceeds the threshold h, and is false otherwise.

stype considers control words one after another until either gm is true or all control words have been considered, in which case the statement is assumed to be an arithmetic-assignment statement. In either case, the value of stype is the value of a specialist function written specifically for one statement type. For example, if gm [p ; (C O M M O N) ; h] is true, the value of stype is the value of ucom[p], a function written specifically to deal with COMMON statements.

There are 34 of these specialist functions for specific statement types, and it is difficult to say anything in general about them, other than the fact that their names begin with the letter "u." However, many of them make use of what amounts to a library of other functions, and these will be described briefly.

2. Utility Functions

head[x;n]: Returns the list of elements one to n from the list x.

---

*In the Appendix all function definitions are given in so-called S-expression notation. For ease in reading, it is much better to use meta-language or M-expression notation. The rules for translating from one to the other are given in LISP 1.5 Programmer's Manual (MIT Press, Cambridge, Massachusetts, 1962).

tail[x;n]: Returns the list of elements n on from the list x.

seg[x;m;n]: Returns that part of the list x strictly between the mth and the nth element.

nel[n;x]: Returns the nth element of the list x.

inside[x]: Returns x stripped of its first and last elements.

alt[p]: Returns the A-list for the P-list p.

3. Utility Predicates

letterp[y]: true iff the atom y is one of the twenty-six letters.

digitp[y]: true iff the atom y is one of the ten numerals.

anp[y]: or[letterp[y] ; digitp[y]].

specialp[y]: true iff the atom y is one of the ten special FORTRAN characters. The following atoms are used to represent these characters:

atom	character
LB	(
ES	=
AS	*
SL	/
PL	+
MI	-
PE	.
CO	,
DS	$
RB	)

signp[y]: or[eq[y;PL] ; eq[y;MI]].

memberfp[y;f]: true iff the atom y appears as one of the characters in the character-confidence pairs in the L-list f.

4. Syntax Predicates

Fourteen predicates have been written for fourteen common FORTRAN forms, such as variable names and expressions. Their arguments

are S-lists, and they are true if and only if the S-list is a syntactically valid form. The predicates are closely related to a family of functions which remove valid forms from the head of an S-list; the names of these functions always begin with the letter "s." Table VI gives the fourteen predicates, the syntactic forms they test, and the functions they call upon.

5. Dynamic-Programming Functions

The two most important functions for dynamic programming are build and peeloff. They correspond to the construction of a graph from a P-list and the retracing of that graph for the string of next lower total confidence, the details of which were described in the Fourth Quarterly Report. To be more specific, build has one argument, a P-list, and creates a list v representing the graph. This list is best described with the aid of an example. If

$$p \leftarrow (((B\ 40)\ (E\ 40)) \\ ((O\ 10)\ (C\ 30)))$$

then

$$v \leftarrow (((80\ (0\ .\ 40))\ (70\ (C\ .\ 40))) \\ ((40\ (B\ .\ 0)\ (E\ .\ 0)))) \quad .$$

In general, if p has n top-level elements, then so does v, but the correspondence between them is in reverse order; v_k, the kth top-level element of v corresponds to the mth top-level element of p, where $m = n - k + 1$. Now v_k is a list whose elements are lists of the following form:

$$(Con_0\ (Char_1\ .\ Con_1)\ (Char_2\ .\ Con_2)\ \ldots\ (Char_j\ .\ Con_j)) \quad .$$

The first element, Con_0, is the cumulative confidence that can be obtained by making selections of characters from the first m-1 lists in p. The following elements are dotted pairs which tell (1) which

Table VI

FORTRAN SYNTAX PREDICATES

Predicate	Form	Functions
expp	expression	sexp
expip	expression-list	sexpl
		scoexplon
	unsigned-expression	suexp
		ssexpon
	term	sterm
	factor	sfactor
	primary	sprimary
		svfon
	bracketed-expression	sbexp
	bracketed-expression-list	sbexpl
		srb
		sfonterm
		seonfactor
	constant	sconstant
		sprefix
		sprodct
		sintegeron
		sindoton
		seon
sintp	signed-integer	ssint
variablep	variable	svar
identifierp	identifier	sid
		ssaon
		sbexplon
bidlistp	bracketed-identifier-list	sbidlist
idlistp	identifier-list	sidlist
		scodimlon
bintlistp	bracketed-integer-list	sbintlist
intlistp	integer-list	sintlist
		scointlistan
	integer-label	sintegerl
integerp	integer	sinteger
inttriplp	integer-triple	stripletriple
		scointpair
intpairp	integer-pair	sintpair
		scoint
dimlistp	dimension-list	sdimlist
		scodimlifton
	bracketed-limit-list	sbrimlist
limlistp	limit-list	slimlist
		scolimiston
	limit	slim
		sslsinton
		ss:

character to select from the mth L-list, and (b) what will be the corresponding cumulative confidence for the $(k+1)$th element of v.

In our example, we see from v_1 that we can obtain cumulative confidences of 80 or 70, and if we wish to obtain 70 we must select "C" and have a cumulative confidence of 40 in v_2. This happens to be the only cumulative confidence in v_2, and it can be achieved by selecting either "B" or "E." In general, one can choose any desired cumulative confidence shown in v_1, and trace back through the other elements of v to find all of the strings of characters from the P-list that will yield this total confidence.

The purpose of _peeloff_ is to allow this searching to be done systematically, yielding strings corresponding to successively lower total confidences. Each time peeloff is called, it returns a list of the form $(s\ t\ w)$, where s is the S-list string of characters, t is the total confidence of that string, and w is a list of decision-history that is needed for the next call of _peeloff_, itself a function of u, v, w. Initially, we take $u = 0$ and $w = NIL$ to obtain the string of highest total confidence. Repeated calls will yield strings of successively lower total confidence until all possibilities are exhausted, at which point _peeloff_ returns NIL. The functions _build_ and _peeloff_ use the following subfunctions:

build	_peeloff_
build1	choose1
addon	choose
putit	itsl
ordered	its
largest	find
delete	cho
	next
	down1
	backup
	rightspot
	findnxt
	belong

The function dyprog[p ; pred] uses build and pecloff. It obtains from p the list of highest total confidence that satisfies the functional-argument predicate "pred." In principle, dyprog can be used in conjunction with the syntax predicates to obtain highest confidence expressions, variables, integer-lists, etc. In practice, it is usually mandatory to do some preprocessing of the P-list before applying dyprog. However, the following functions are available that use dyprog directly:

$$\underline{\text{dexp}}[p] \leftarrow \text{dyprog}[p;\text{function}[\text{expp}]]$$

$$\underline{\text{dintlist}}[p] \leftarrow \text{dyprog}[p;\text{function}[\text{bintlistp}]]$$

$$\underline{\text{dintlist}}[p] \leftarrow \text{dyprog}[p;\text{function}[\text{intlistp}]] \quad .$$

6. Functions for Resolving Forms

Special functions have been written for the following FORTRAN forms: scalar variable, integer-list, bracketed-integer-list, and expression. These functions try to produce a high-confidence, valid FORTRAN form from a P-list. If they succeed, they return the form as an S-list; otherwise they return NIL.

svalvar[p]: returns the highest confidence variable name that can be obtained from p. After checking that the length of p is between one and eight, it uses startvar to select the highest confidence letter to begin the name and restvar and restvar1 to select the highest confidence string of alphanumerics to complete the name.

intlist[p]: returns the highest confidence list of integer labels, integers between 1 and 99999 separated by commas. It effectively preprocesses the P-list for dintlist, the dynamic programming version of intlist. This preprocessing accomplishes three things:

(1) The P-list is checked to make sure that each L-list contains at least one digit or comma (digcopp, digcolp, digcop).

(2) All choices other than the highest confidence digit and comma are removed from each L-list (intlist, digco, combine, coon, digon).

(3) The highest confidence digits are selected for the first and the last L-lists. The resulting P-list is given to dintlist to obtain the final answer (intlist1, pickdigit).

bintlist[p] - numlist[p]: returns the bracketed list of integer-labels having highest total confidence. It merely checks to make sure that the first and last L-lists in p contain left and right brackets as possible characters, and then passes the main problem to intlist.

expn[p]: uses heuristics to break an expression into subexpressions small enough to be treated by dynamic programming. The basic strategy is to select some of the operators +, -, *, and / that appear as possible characters in the P-list and use them to break the P list into segments that are simpler expressions. If these candidate subexpressions are sufficiently simple, i.e., sufficiently short, the dynamic programming function exp1 desp is used to resolve them. If all of the segments can be successfully resolved, the resulting S lists are strung together, separated by the appropriate operators, to yield the final answer. If any segment can not be resolved, another selection of operators is made and the process is repeated.

The function pplab returns a list of numbers locating the elements of the P list that are possible operators. If there are n such elements, a binary n-tuple "upto" is used to specify a particular selection. This n-tuple is initially set by init to be all ones. Each successive selection is obtained by decmod2, which subtracts one from the binary number "upto," and a list locating the operators selected is produced by dbd. Thus all possible combinations of operators and non operators are tried, starting with selecting all operators. This procedure tends to break the P list into many short segments at first, postponing the consideration of longer segments which can not be handled as well by dynamic programming. The maximum length list ever given to the dynamic programming functions is fixed by the function lll.

D. Specialist Functions

Only 23 of the 34 specialist functions have been written, but these include most of the most common types of FORTRAN statements. Their operation will be described briefly.

1. Declaration Statements

ucom[p] (COMMON): applies idlist (temporarily available using dyprog and idlistp) to the tail of p that follows the control word. Like most of the specialist functions to be described, ucom begins by removing spaces with despace. If the tail is successfully resolved, it returns an S-list consisting of the control word followed by the resolved tail; otherwise it returns an error message.

udim[p] (DIMENSION): applies dimlist (temporarily available using dyprog and dimlistp) to the tail of p following the control word.

ufn[p] (FUNCTION): looks for the first possible left bracket to break the P-list into a name followed by a bracketed-identifier-list. If the resolution is successful, the S-lists are strung together and returned; if not, the next possible left bracket is sought and the process is repeated.

usubrtn[p] (SUBROUTINE): uses the same logic as ufn.

2. Computation Statements

uas[p] (ARITHMETIC ASSIGNMENT): looks for the first possible equal sign to break the P-list into a scalar variable followed by an equal sign and an expression. If the resolution is successful, the S-lists are strung together and returned; if not, the next possible equal sign is sought and the process is repeated.

ucall[p] (CALL): returns scalvr[p] if the result is not NIL; if it is, ucall uses basically the same logic as ufn in trying to split the P-list into a name followed by a bracketed expression list.

3. Control Statements

ugo[p], ugoto[p] (GO TO): If the average confidence that the tail of the P-list is a number exceeds the threshold H3, then the

statement is assumed to be an ordinary GO TO statement, and the highest confidence integer is selected for the tail. If not, but if a possible left bracket follows the control word, and a possible comma and the right bracket appear elsewhere in the P-list, then the statement is assumed to be a COMPUTED GO TO statement, which is split into a bracketed-integer-list and an expression by negt. If neither of these conditions is satisfied, then the statement is assumed to be an ASSIGNED GO TO statement, and the highest confidence name is selected for the tail.

ado[p] (DO): looks for the first possible equal sign to split the tail of the P-list following the control word into an integer-identifier followed by either two or three expressions separated by commas. frontdo resolves the left half and backdo resolves the right half. Failure at any step causes the function to look for the next possible equal sign and to repeat the process.

aifa[p] , aifi[p] (IF): tries to split the tail into a bracketed expression (which is itself an expression) followed by an integer-triple. It works back from the end of the A list, looking for a possible digit-comma-digit sequence. If successful, it looks for another such sequence, and if successful again it looks for a possible right bracket-digit sequence. If found, this is used as the tentative split, and iflist (temporarily available using dypreg and intriplep) is applied to the right half of the P-list. If this fails, the next possible right bracket-digit sequence is sought for another tentative split, and the process is repeated. If iflist succeeds, exp is applied to the left half, and if this also succeeds the separate A lists are strung together and returned.

assaigp[p] (ASSIGN): looks for the control word "TO" to split the tail into a variable followed by an integer. If a tentative control word is found, numl is applied to resolve the right half and var is applied to resolve the left half.

apause[p] (PAUSE): applies numl to the tail of p following the control word.

4. Input/Output Statements

The input/output statements that do not concern magnetic tape are identical except for the control word. For those statements, one function, genio, is used to resolve the tail of p following the control word. genio looks for a comma to split the tail into an integer followed by an input-output-list, applying numl to the left half and iolist (temporarily available using dyprog and idlistp) to the right half. The following input/output statements are handled in this way:

uacc[p]	(ACCEPT)
upnch[p]	(PUNCH)
upntp[p]	(PUNCH TAPE)
uread[p]	(READ)
utype[p]	(TYPE)
uacctp[p]	(ACCEPT TAPE)
uprint[p]	(PRINT) .

Three other statements concerning magnetic tape handling also have identical forms except for the control word, and only require that exp be applied to the tail of the p-list. The corresponding functions are:

uef[p]	(END FILE)
urwnd[p]	(REWIND)
ubsp[p]	(BACKSPACE) .

APPENDIX

LISTING OF THE SYMBOLIC FILES FOR THE SYNTAX ANALYSIS PROGRAM

```
(PRIN1 (QUOTE THIS$ FILE WAS CREATED ON $))
(PRINT (QUOTE 28-JUNE-1967))
(DEFINEQ

(SPACE
  (LAMBDA (P)
    (COND
      ((NOT (EQUAL SP (CAAAR P)))
        (CLPSE P))
      (T (SPACE (CDR P))))))

(CLPSE
  (LAMBDA (P)
    (COND
      ((NULL (CDR P))
        P)
      ((AND (EQUAL SP (CAAAR P))
            (EQUAL SP (CAAADR P)))
        (CLPSE (CDR P)))
      (T (CONS (CAR P)
            (CLPSE (CDR P)))))))

(NFL
  (LAMBDA (N L)
    (PROG (M U)
          (SETQ M 1)
          (SETQ U L)
      LOOP(COND
            ((EQUAL M N)
              (RETURN (CAR U))))
          (SETQ M (ADD1 M))
          (SETQ U (CDR U))
          (GO LOOP)
      )))

(MATCH2
  (LAMBDA (L X)
    (COND
      ((NULL L)
        0)
      ((EQUAL X (CAAR L))
        (CADAR L))
      (T (MATCH2 (CDR L)
            X)))))

(MATCH
  (LAMBDA (P M)
    (PROG (P1 M1 C D)
          (SETQ C 0)
          (SETQ D M)
          (SETQ P1 P)
          (SETQ M1 M)
      A   (COND
            ((NULL M1)
```

```
                (RETURN (QUOTIENT C (LENGTH D)))))
            (SETQ C (PLUS C (MATCH2 (CAR P1)
                            (CAR M1))))
            (SETQ P1 (CDR P1))
            (SETQ M1 (CDR M1))
            (GO A)
        )))

(GM
  (LAMBDA (P M M)
    (COND
      ((AND (GREATERP (ADD1 (LENGTH P))
                      (LENGTH M))
            (GREATERP (MATCH P M)
                      M))
       T)
      (T NIL))))

(STYPE
  (LAMBDA (P)
    (COND
      ((AND (EQUAL 6 (LENGTH P))
            (GM P (QUOTE (R E T U R N))
                M1))
       (QUOTE (RETURN)))
      ((AND (EQUAL 4 (LENGTH P))
            (GM P (QUOTE (S T O P))
                M1))
       (QUOTE (STOP)))
      ((AND (EQUAL 5 (LENGTH P))
            (GM P (QUOTE (P A U S E))
                M1))
       (QUOTE (PAUSE)))
      ((AND (EQUAL 8 (LENGTH P))
            (GM P (QUOTE (C O N T I N U E))
                M1))
       (QUOTE (CONTINUE)))
      ((AND (EQUAL 3 (LENGTH P))
            (GM P (QUOTE (E N D))
                M1))
       (QUOTE (END)))
      ((GM P (QUOTE (G O SP T O SP))
           M)
       (UGO P))
      ((AND (GM P (QUOTE (G O SP))
                M)
            (GREATERP (NUMVAL (NEL 4 P))
                      M2))
       (UDO P))
      ((GM P (QUOTE (G O T O))
           M)
       (UGOTO P))
      ((GM P (QUOTE (C A L L))
           M)
```

```
   (UCALL P))
((GM P (QUOTE (S U B R O U T I N E))
     H)
   (USBRTN P))
((GM P (QUOTE (D I M E N S I O N))
     H)
   (UDIM P))
((GM P (QUOTE (P A U S E))
     H)
   (UPAUSE P))
((GM P (QUOTE (C O M M O N))
     H)
   (UCOM P))
((GM P (QUOTE (F O R M A T))
     H)
   (UFORM P))
((GM P (QUOTE (A C C E P T SP T A P E))
     H)
   (UACCTP P))
((GM P (QUOTE (I F LB S E N S E SP S W I T C H))
     H)
   (UIFSS P))
((GM P (QUOTE (A S S I G N))
     H)
   (UASSIGN P))
((GM P (QUOTE (A C C E P T))
     H)
   (UACC P))
((GM P (QUOTE (E N D SP F I L E))
     H)
   (UEF P))
((GM P (QUOTE (F U N C T I O N))
     H)
   (UFN P))
((GM P (QUOTE (E Q U I V A L E N C E))
     H)
   (UEQV P))
((GM P (QUOTE (P U N C H SP T A P E))
     H)
   (UPNTP P))
((GM P (QUOTE (P U N C H))
     H)
   (UPNCH P))
((GM P (QUOTE (T Y P E))
     H)
   (UTYPE P))
((GM P (QUOTE (R E A D SP T A P E))
     H)
   (URDTP P))
((GM P (QUOTE (R E A D SP I N P U T SP T A P E))
     H)
   (URIT P))
((GM P (QUOTE (R E A D))
     H)
```

```
        (UREAD P))
       ((GM P (QUOTE (R E W I N D))
          H)
        (URWND P))
       ((GM P (QUOTE (W R I T E SP T A P E))
          H)
        (UWT P))
       ((GM P (QUOTE (P R I N T))
          H)
        (UPRNT P))
       ((GM P (QUOTE (S E N S E SP L I G H T))
          H)
        (USL P))
       ((GM P (QUOTE (B A C K S P A C E))
          H)
        (UBSP P))
       ((GM P (QUOTE (I F SP F L O A T I N G SP O V E R F L O
W))
          H)
        (UIFFO P))
       ((GM P (QUOTE (W R I T E SP O U T P U T SP T A P E))
          H)
        (UWOT P))
       ((GM P (QUOTE (I F LB S E N S E SP L I G H T))
          H)
        (UFISL P))
       ((GM P (QUOTE (I F SP))
          H)
        (UIFS P))
       ((GM P (QUOTE (I F LB))
          H)
        (UIFL P))
       (T (UAS (DESPACE P))))))

(ST
  (LAMBDA (P)
    (STYPE (SPACE P))))

(NUMVAL
  (LAMBDA (L)
    (COND
      ((NULL L)
        0)
      ((NUMBERP (CAAR L))
        (CADAR L))
      (T (NUMVAL (CDR L))))))

(ALT1
  (LAMBDA (L)
    (MAPCAR L (FUNCTION CAR))))

(ALT
  (LAMBDA (P)
    (MAPCAR P (FUNCTION ALT1))))
```

```
(NUM2
  (LAMBDA (K)
    (COND
      ((NULL K)
        NIL)
      ((NUMBERP (CAR K))
        (CAR K))
      (T (NUM2 (CDR K))))))

(NUM3
  (LAMBDA (A)
    (MAPCAR A (FUNCTION NUM2))))

(NUM1
  (LAMBDA (A)
    (COND
      ((MEMBER NIL (NUM3 A))
        NIL)
      (T (NUM3 A)))))

(UGO1
  (LAMBDA (P N)
    (COND
      ((EQUAL NIL (NUM1 (ALT (TAIL P N))))
        (QUOTE (MISSING NUMBERS IN GO TO STATEMENT)))
      (T (COND
           ((EQUAL 7 N)
             (APPEND (QUOTE (G O SP T O SP))
               (NUM1 (ALT (TAIL P 7)))))
           ((EQUAL 6 N)
             (APPEND (QUOTE (G O T O SP))
               (NUM1 (ALT (TAIL P 6))))))))))

(NMTCH
  (LAMBDA (P)
    (PROG (C D)
          (SETQ C 0)
          (SETQ D (LENGTH P))
      A   (COND
            ((NULL P)
              (RETURN (QUOTIENT C D))))
          (SETQ C (PLUS C (NUMVAL (CAR P))))
          (SETQ P (CDR P))
          (GO A)
      ))))

(MEM2
  (LAMBDA (X A)
    (COND
      ((NULL A)
        NIL)
      ((MEMBER X (CAR A))
        T)
```

```
        (T (MEM2 X (CDR A))))))

(UGO
  (LAMBDA (P)
    (UGOX P 7)))

(UGOTO
  (LAMBDA (P)
    (UGOX P 6)))

(UGOX
  (LAMBDA (P N)
    ((LAMBDA (X)
        (COND
          ((AND (GREATERP (NMTCH X)
                  H3)
                (LESSP (LENGTH X)
                  6)
                (ALLNUMP X))
            (UGO1 P N))
          ((AND (MEMBER LB (ALT1 (NEL N P)))
                (MEM2 CO (ALT X))
                (MEM2 RB (ALT X)))
            (UCGT P N))
          (T (UASGT P N))))
      (TAIL P N))))

(ALLNUMP
  (LAMBDA (P)
    ((LAMBDA (A)
        (COND
          ((NULL (NUM1 A))
            NIL)
          (T T)))
      (ALT P))))

(HEAD
  (LAMBDA (P N)
    (HEAD1 (CDR P)
      N
      (LIST (CAR P)))))

(HEAD1
  (LAMBDA (P C U)
    (COND
      ((NULL P)
        (REVERSE U))
      ((EQUAL C 1)
        (REVERSE U))
      (T (HEAD1 (CDR P)
          (SUB1 C)
          (CONS (CAR P)
            U))))))
```

```
(TAIL
  (LAMBDA (P N)
    (COND
      ((NULL P)
        NIL)
      ((EQUAL N 1)
        P)
      (T (TAIL (CDR P)
          (SUB1 N))))))

(NUMEM
  (LAMBDA (A)
    (COND
      ((NULL A)
        NIL)
      ((NUMBERP (CAR A))
        T)
      (T (NUMEM (CDR A))))))

(UPAUSE
  (LAMBDA (P)
    (COND
      ((NULL (NUM1 (ALT (TAIL P 7))))
        (QUOTE (MISSING NUMBER IN PAUSE STATEMENT)))
      (T (APPEND (QUOTE (P A U S E SP))
          (NUM1 (ALT (TAIL P 7)))))))))

(UASGT
  (LAMBDA (P N)
    (COND
      ((NULL (VAR (TAIL P N)))
        (QUOTE (NO VARIABLE IN ASSIGNED GO TO)))
      (T (APPEND (QUOTE (G O SP T O SP))
          (VAR (TAIL P N)))))))

(UCGT
  (LAMBDA (P N)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE COMPUTED GO TO)))
          (T (APPEND (QUOTE (G O SP T O SP))
              X))))
      (UCGT2 (TAIL P N)))))

(UCGT2
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V Q)
              (SETQ C 4)
          LOOP(COND
                ((LESSP (LENGTH A)
                    5)
                  (RETURN NIL))
```

```
                ((NOT (AND (NUMEM (CADR A))
                           (MEMBER RB (CADDR A))
                           (OR (MEMBER SP (CADDDR A))
                               (MEMBER CO (CADDDR A)))))
                  (GO B)))
               (SETQ U (BINTLIST (DESPACE (HEAD P (SUB1 C)))))
)
               (COND
                 ((NULL U)
                   (GO B)))
               (SETQ V (EXP (DESPACE (TAIL P (ADD1 C)))))
               (COND
                 ((NULL V)
                   (GO B)))
               (SETQ Q (APPEND (QUOTE (SP))
                   V))
               (RETURN (APPEND U Q))
          B    (SETQ C (ADD1 C))
               (SETQ A (CDR A))
               (GO LOOP)
          ))
       (ALT P))))

(DESPACE
  (LAMBDA (P)
    (COND
      ((NULL P)
        NIL)
      ((EQUAL SP (CAAAR P))
        (DESPACE (CDR P)))
      (T (CONS (CAR P)
          (DESPACE (CDR P)))))))

(UIFS
  (LAMBDA (P)
    ((LAMBDA (A)
        (COND
          ((MEMBER SP (CADDR A))
            (UIFS1 (UIF (DESPACE (CDDDR P)))))
          ((MEMBER LB (CADDR A))
            (UIFL P))
          (T (QUOTE (CANT RESOLVE IF STATEMENT)))))
       (ALT P))))

(UIFL
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE IF STATEMENT)))
          (T (APPEND (QUOTE (T F))
              X))))
       (UIF (DESPACE (CDDR P))))))
```

```
(UIFS1
  (LAMBDA (X)
    (COND
      ((NULL X)
        (QUOTE (CANT RESOLVE IF STATEMENT)))
      (T (APPEND (QUOTE (I F SP))
          X)))))

(UIF
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V)
              (SETQ C 1)
          F   (COND
                ((LESSP (LENGTH A)
                    7)
                  (RETURN NIL))
                ((NOT (AND (NUMEM (CAR A))
                           (MEMBER CO (CADR A))
                           (NUMEM (CADDR A))))
                  (GO B)))
              (SETQ C (PLUS C 2))
              (SETQ A (CDDR A))
          E   (COND
                ((LESSP (LENGTH A)
                    5)
                  (RETURN NIL))
                ((NOT (AND (NUMEM (CAR A))
                           (MEMBER CO (CADR A))
                           (NUMEM (CADDR A))))
                  (GO D)))
          H   (COND
                ((LESSP (LENGTH A)
                    4)
                  (RETURN NIL))
                ((NOT (AND (NUMEM (CAR A))
                           (MEMBER RB (CADR A))))
                  (GO G)))
              (SETQ U (IFLIST (REVERSE (HEAD (REVERSE P)
                      C))))
              (COND
                ((NULL U)
                  (GO G)))
              (SETQ V (EXP (REVERSE (TAIL (REVERSE P)
                      (ADD1 C)))))
              (COND
                ((NULL V)
                  (GO G)))
              (RETURN (APPEND V U))
          G   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
              (GO H)
          D   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
```

```
                (GO E)
            B   (SETQ C (ADD1 C))
                (SETQ A (CDR A))
                (GO F)
            ))
        (REVERSE (ALT P)))))

(UDO
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE DO STATEMENT)))
          (T (APPEND (QUOTE (D O SP))
                X))))
      (UDO1 (CDDDR P)))))

(UDO1
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C O U V)
              (SETQ C 4)
          LOOP(COND
                ((LESSP (LENGTH A)
                    7)
                  (RETURN NIL))
                ((NOT (MEMBER (CADDDR A)))
                  (GO B)))
              (SETQ U (FRONTDO (HEAD P (SUB1 C))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (BACKDO (DESPACE (TAIL P (ADD1 C)))))
              (COND
                ((NULL V)
                  (GO B)))
              (SETQ O (APPEND (QUOTE (ES))
                  V))
              (RETURN (APPEND U O))
          B   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
              (GO LOOP)
          ))
      (ALT P))))

(FRONTDO
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C O U V A1)
              (SETQ C 2)
              (SETQ A1 A)
          LOOP(COND
                ((LESSP (LENGTH A)
                    2)
```

```
                    (RETURN NIL))
                  ((NOT (MEMBER SP (CADR A)))
                    (GO B)))
                (SETQ U (NUM1 (HEAD A1 (SUB1 C))))
                (COND
                  ((NULL U)
                    (GO B)))
                (SETQ V (IDENT (DESPACE (TAIL P (ADD1 C)))))
                (COND
                  ((NULL V)
                    (GO B)))
                (SETQ Q (APPEND (QUOTE (SP))
                    V))
                (RETURN (APPEND U Q))
          B     (SETQ C (ADD1 C))
                (SETQ A (CDR A))
                (GO LOOP)
          ))
      (ALT P))))

(BACKDO
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C Q U V)
              (SETQ C 2)
          TAG1(COND
                ((LESSP (LENGTH A)
                    2)
                  (RETURN (BACKDO1 P)))
                ((NOT (MEMBER C0 (CADR A)))
                  (GO B)))
              (SETQ U (EXP (HEAD P (SUB1 C))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (EXP (TAIL P (ADD1 C))))
              (COND
                ((NULL V)
                  (GO B)))
              (SETQ Q (APPEND (QUOTE (C0))
                  V))
              (RETURN (APPEND U Q))
          B   (SETQ A (CDR A))
              (SETQ C (ADD1 C))
              (GO TAG1)
          ))
      (ALT P))))

(BACKDO1
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V Q)
              (SETQ C 2)
          F   (COND
```

```
                    ((LESSP (LENGTH A)
                            3)
                      (RETURN NIL))
                    ((NOT (MEMBER CO (CADR A)))
                      (GO B)))
                  (SETQ U (EXP (HEAD P (SUB1 C))))
                  (COND
                    ((NULL U)
                      (GO B)))
                  (SETQ V (BACKDO2 (TAIL P (ADD1 C))))
                  (COND
                    ((NULL V)
                      (GO B)))
                  (SETQ Q (APPEND (QUOTE (CO))
                                  V))
                  (RETURN (APPEND U Q))
             B    (SETQ A (CDR A))
                  (SETQ C (ADD1 C))
                  (GO F)
             ))
        (ALT P))))

(BACKDO2
  (LAMBDA (P)
    ((LAMBDA (A)
       (PROG (C U V Q)
             (SETQ C 2)
         F   (COND
               ((LESSP (LENGTH A)
                       3)
                 (RETURN NIL))
               ((NOT (MEMBER CO (CADR A)))
                 (GO B)))
             (SETQ U (EXP (HEAD P (SUB1 C))))
             (COND
               ((NULL U)
                 (GO B)))
             (SETQ V (EXP (TAIL P (ADD1 C))))
             (COND
               ((NULL V)
                 (GO B)))
             (SETQ Q (APPEND (QUOTE (CO))
                             V))
             (RETURN (APPEND U Q))
         B   (SETQ A (CDR A))
             (SETQ C (ADD1 C))
             (GO F)
         ))
      (ALT P))))

(UAS
  (LAMBDA (P)
    ((LAMBDA (A)
       (PROG (C U V Q)
```

```
                (SETQ C 2)
           LOOP(COND
                 ((LESSP (LENGTH A)
                         3)
                   (RETURN (QUOTE (CANT RESOLVE ARITHMETIC ASSIGNMENT
 STATEMENT))))
                 ((NOT (MEMBER ES (CADR A)))
                   (GO B)))
                (SETQ U (VAR (HEAD P (SUB1 C))))
                (COND
                  ((NULL U)
                    (GO B)))
                (SETQ V (EXPH (TAIL P (ADD1 C))))
                (COND
                  ((NULL V)
                    (GO B)))
                (SETQ Q (APPEND (QUOTE (ES))
                    V))
                (RETURN (APPEND U Q))
           B    (SETQ C (ADD1 C))
                (SETQ A (CDR A))
                (GO LOOP)
            ))
       (ALT P))))

(UASSIGN
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE ASSIGN STATEMENT)))
          (T (APPEND (QUOTE (A S S I G N SP))
                X))))
      (UASSIGN1 (TAIL P 8)))))

(UASSIGN1
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V Q A1)
              (SETQ C 3)
              (SETQ A1 A)
          LOOP(COND
                ((LESSP (LENGTH A)
                        3)
                  (RETURN NIL))
                ((NOT (AND (MEMBER T (NEL 3 A))
                           (MEMBER Q (NEL 4 A))))
                  (GO B)))
              (SETQ U (NUM1 (HEAD A1 (DIFFERENCE C 2))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (VAR (TAIL P (PLUS C 3))))
              (COND
```

```
                ((NULL V)
                  (GO B)))
               (SETQ Q (APPEND (QUOTE (SP T O SP))
                  V))
               (RETURN (APPEND U Q))
          B    (SETQ C (ADD1 C))
               (SETQ A (CDR A))
               (GO LOOP)
          ))
       (ALT P))))

(UFN
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE FUNCTION STATEMENT)))
         (T (APPEND (QUOTE (F U N C T I O N SP))
            X))))
      (UFN1 (DESPACE (TAIL P 10))))))

(UFN1
  (LAMBDA (P)
    ((LAMBDA (A)
       (PROG (C U V)
             (SETQ C 2)
         LOOP(COND
               ((LESSP (LENGTH A)
                   4)
                 (RETURN NIL))
               ((NOT (MEMBER LB (CADR A)))
                 (GO B)))
             (SETQ U (IDENT (HEAD P (SUB1 C))))
             (COND
               ((NULL U)
                 (GO B)))
             (SETQ V (BIDLIST (TAIL P C)))
             (COND
               ((NULL V)
                 (GO B)))
             (RETURN (APPEND U V))
          B  (SETQ C (ADD1 C))
             (SETQ A (CDR A))
             (GO LOOP)
          ))
      (ALT P))))

(UCALL
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE CALL STATEMENT)))
         (T (APPEND (QUOTE (C A L L SP))
```

```
                X))))
        (UCALL1 (DESPACE (TAIL P 6))))))

(UCALL1
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (UCALL2 P))
          (T X)))
      (IDENT P))))

(UCALL2
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V)
              (SETQ C 2)
          LOOP(COND
                ((LESSP (LENGTH A)
                        4)
                  (RETURN NIL))
                ((NOT (MEMBER LB (CADR A)))
                  (GO B)))
              (SETQ U (IDENT (HEAD P (SUB1 C))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (EXPL (TAIL P C)))
              (COND
                ((NULL V)
                  (GO B)))
              (RETURN (APPEND U V))
          B   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
              (GO LOOP)
        ))
      (ALT P))))

(USBRTN
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE SUBROUTINE STATEMENT)))
          (T (APPEND (QUOTE (S U B R O U T I N E SP))
                X))))
      (USBRTN1 (DESPACE (TAIL P 12))))))

(USBRTN1
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (USBRTN2 P))
```

```
          (T X)))
       (IDENT P))))

(USBRTN2
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V)
              (SETQ C 2)
          LOOP(COND
                ((LESSP (LENGTH A)
                        4)
                  (RETURN NIL))
                ((NOT (MEMBER LB (CADR A)))
                  (GO B)))
              (SETQ U (IDENT (HEAD P (SUB1 C))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (BIDLIST (TAIL P C)))
              (COND
                ((NULL V)
                  (GO B)))
              (RETURN (APPEND U V))
          B   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
              (GO LOOP)
        ))
      (ALT P))))

(UCOM
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE COMMON STATEMENT)))
          (T (APPEND (QUOTE (C O M M O N SP))
                     X))))
      (IDLIST (DESPACE (TAIL P 8))))))

(UDIM
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE DIMENSION STATEMENT)))
          (T (APPEND (QUOTE (D I M E N S I O N SP))
                     X))))
      (DIMLIST (DESPACE (TAIL P 11))))))

(UACC
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
```

```
            (QUOTE (CANT RESOLVE ACCEPT STATEMENT)))
          (T (APPEND (QUOTE (A C C E P T SP))
                     X))))
      (GENIO (TAIL P 8)))))

(UPNCH
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE PUNCH STATEMENT)))
         (T (APPEND (QUOTE (P U N C H SP))
                    X))))
      (GENIO (TAIL P 7)))))

(UPNTP
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE PUNCH TAPE STATEMENT)))
         (T (APPEND (QUOTE (P U N C H SP T A P E SP))
                    X))))
      (GENIO (TAIL P 12)))))

(UREAD
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE READ STATEMENT)))
         (T (APPEND (QUOTE (R E A D SP))
                    X))))
      (GENIO (TAIL P 6)))))

(UTYPE
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE TYPE STATEMENT)))
         (T (APPEND (QUOTE (T Y P E SP))
                    X))))
      (GENIO (TAIL P 6)))))

(UACCTP
  (LAMBDA (P)
    ((LAMBDA (X)
       (COND
         ((NULL X)
           (QUOTE (CANT RESOLVE ACCEPT TAPE STATEMENT)))
         (T (APPEND (QUOTE (A C C E P T SP T A P E SP))
                    X))))
      (GENIO (TAIL P 13)))))
```

```
(UPRNT
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE PRINT STATEMENT)))
          (T (APPEND (QUOTE (P R I N T SP))
                X))))
      (GENIO (TAIL P 7)))))

(GENIO
  (LAMBDA (P)
    ((LAMBDA (A)
        (PROG (C U V O A1)
              (SETQ C 2)
              (SETQ A1 A)
          LOOP(COND
                ((OR (LESSP (LENGTH A)
                            3)
                     (GREATERP C 6))
                  (RETURN NIL))
                ((NOT (AND (NUMEM (CAR A))
                           (MEMBER CO (CADR A))))
                  (GO B)))
              (SETQ U (NUM1 (HEAD A1 (SUB1 C))))
              (COND
                ((NULL U)
                  (GO B)))
              (SETQ V (IOLIST (TAIL (DESPACE P)
                                    (ADD1 C))))
              (COND
                ((NULL V)
                  (GO B)))
              (SETQ O (APPEND (QUOTE (CO))
                              V))
              (RETURN (APPEND U O))
          B   (SETQ C (ADD1 C))
              (SETQ A (CDR A))
              (GO LOOP)
          ))
      (ALT (DESPACE P)))))

(UFF
  (LAMBDA (P)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            (QUOTE (CANT RESOLVE END FILE STATEMENT)))
          (T (APPEND (QUOTE (E N D SP F I L E))
                X))))
      (EXP (DESPACE (TAIL P 10))))))

(URWND
```

```
    (LAMBDA (P)
      ((LAMBDA (X)
           (COND
             ((NULL X)
               (QUOTE (CANT RESOLVE REWIND STATEMENT)))
             (T (APPEND (QUOTE (R E W I N D SP))
                  X))))
        (EXP (DESPACE (TAIL P 8))))))

 (URSP
   (LAMBDA (P)
     ((LAMBDA (X)
          (COND
            ((NULL X)
              (QUOTE (CANT RESOLVE BACKSPACE STATEMENT)))
            (T (APPEND (QUOTE (B A C K S P A C E SP))
                 X))))
       (EXP (DESPACE (TAIL P 11))))))

 (INIT
   (LAMBDA (N)
     (INIT1 N NIL)))

 (INIT1
   (LAMBDA (N U)
     (COND
       ((EQUAL N 0)
         U)
       (T (INIT1 (SUB1 N)
            (CONS 1 U))))))

 (DECMOD2
   (LAMBDA (N)
     (REVERSE (DECMOD21 (REVERSE N)))))

 (DECMOD21
   (LAMBDA (M)
     (COND
       ((NOT (MEMBER 1 M))
         NIL)
       ((EQUAL 1 (CAR M))
         (CONS 0 (CDR M)))
       (T (CONS 1 (DECMOD21 (CDR M)))))))

 (PMLAB
   (LAMBDA (P PMLIST C)
     ((LAMBDA (X)
          (COND
            ((NULL P)
              NIL)
            ((NULL (CDR P))
              (REVERSE PMLIST))
            ((OR (MEMBER PL X)
                 (MEMBER MI X)
```

```
                (MEMBER AS X)
                (MEMBER SL X))
              (PMLAB (CDR P)
                (CONS C PMLIST)
                (ADD1 C)))
            (T (PMLAB (CDR P)
                PMLIST
                (ADD1 C)))))
        (ALT1 (CAR P)))))

(SEG
  (LAMBDA (X M N)
    (COND
      ((EQUAL M (SUB1 N))
        NIL)
      ((GREATERP N (LENGTH X))
        (TAIL X (ADD1 M)))
      (T (HEAD (TAIL X (ADD1 M))
          (DIFFERENCE (SUB1 N)
            M))))))

(EXPH5
  (LAMBDA (P CD U V)
    (COND
      ((NULL V)
        U)
      (T (EXPH5 P (CDR CD)
          (CONC U (OP (NEL (CAR CD)
                P))
            (CAR V))
          (CDR V))))))

(OP
  (LAMBDA (L)
    (COND
      ((NULL L)
        (QUOTE (ERROR)))
      ((OR (EQUAL (CAAR L)
            PL)
          (EQUAL (CAAR L)
            MI)
          (EQUAL (CAAR L)
            AS)
          (EQUAL (CAAR L)
            SL))
        (LIST (CAAR L)))
      (T (OP (CDR L))))))

(EXPH3
  (LAMBDA (P CD)
    ((LAMBDA (X)
        (COND
          ((NULL X)
            NIL)
```

```
          (T (EXPH5 P CD (CAR X)
                (CDR X)))))
      (EXPH4 NIL (EXPH41 P (CONS 0 (APPEND CD (QUOTE (1000))
))
          NIL)))))

(EXPH4
  (LAMBDA (U V)
    (PROG (R)
      LOOP(COND
            ((NULL V)
              (RETURN (REVERSE U)))
            ((GREATERP (LENGTH (CAR V))
                HE)
              (RETURN NIL)))
          (COND
            ((NULL (CAR V))
              (RETURN NIL)))
          (SETQ R (EXP (CAR V)))
          (COND
            ((NULL R)
              (RETURN NIL)))
          (SETQ U (CONS R U))
          (SETQ V (CDR V))
          (GO LOOP)
      )))

(EXPH41
  (LAMBDA (P CDA W)
    (COND
      ((NULL (CDR CDA))
        (REVERSE W))
      (T (EXPH41 P (CDR CDA)
          (CONS (SEG P (CAR CDA)
              (CADR CDA))
            W))))))

(EXPH2
  (LAMBDA (P PMLIST UPTO)
    (EXPH3 P (DBD NIL PMLIST UPTO))))

(DBD
  (LAMBDA (U PMLIST UPTO)
    (COND
      ((NULL UPTO)
        (REVERSE U))
      ((EQUAL 1 (CAR UPTO))
        (DBD (CONS (CAR PMLIST)
            U)
          (CDR PMLIST)
          (CDR UPTO)))
      (T (DBD U (CDR PMLIST)
          (CDR UPTO))))))
```

```
(EXPH
  (LAMBDA (P)
    (COND
      ((NULL (CDR P))
        (EXP P))
      (T ((LAMBDA (PMLIST)
            (COND
              ((LESSP (LENGTH P)
                  HE)
                (EXP P))
              ((NULL PMLIST)
                NIL)
              (T (EXPH1 P PMLIST (INIT (LENGTH PMLIST))))))
          (PMLAB (CDR P)
            NIL
            2))))))

(EXPH1
  (LAMBDA (P PMLIST UPTO)
    (COND
      ((NULL UPTO)
        NIL)
      (T ((LAMBDA (X)
            (COND
              ((NULL (EXPP X))
                (EXPH1 P PMLIST (DECMOD2 UPTO)))
              (T X)))
          (EXPH2 P PMLIST UPTO))))))

(LETTERP
  (LAMBDA (A)
    (MEMBER A LETTERLIST)))

(DIGITP
  (LAMBDA (A)
    (MEMBER A DIGITLIST)))

(SPECIALP
  (LAMBDA (A)
    (MEMBER A SPECIALLIST)))

(ANP
  (LAMBDA (A)
    (OR (LETTERP A)
      (DIGITP A))))

(SIGNP
  (LAMBDA (A)
    (OR (EQ A PL)
      (EQ A MI))))

(EXPP
  (LAMBDA (L)
    (NULL (SEXP L))))
```

```
(EXPLP
  (LAMBDA (L)
    (NULL (SEXPL L))))

(SEXPL
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      (T (SCOEXPLON (SEXP L))))))

(SCOEXPLON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((EQ (CAR L)
           CO)
        (SEXPL (CDR L)))
      (T L))))

(SEXP
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((SIGNP (CAR L))
        (SUEXP (CDR L)))
      (T (SUEXP L)))))

(SUEXP
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      (T (SSEXPON (STERM L))))))

(SSEXPON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((SIGNP (CAR L))
        (SUEXP (CDR L)))
      (T L))))

(STERM
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      (T (SFONTERM (SFACTOR L))))))
```

```
(SFACTOR
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      (T (SEONFACTOR (SPRIMARY L))))))

(SPRIMARY
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((LETTERP (CAR L))
        (SVFON (CDR L)))
      ((OR (DIGITP (CAR L))
           (EQ (CAR L)
             PE))
        (SCONSTANT L))
      ((EQ (CAR L)
           LB)
        (SBEXP L))
      (T ERRL))))

(SVFON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((ANP (CAR L))
        (SVFON (CDR L)))
      ((EQ (CAR L)
           LB)
        (SBEXPL L))
      (T L))))

(SBEXP
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((EQ (CAR L)
           LB)
        (SRB (SEXP (CDR L))))
      (T ERRL))))

(SBEXPL
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((EQ (CAR L)
           LB)
        (SRB (SEXPL (CDR L))))
      (T ERRL))))
```

```
(SRB
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((EQ (CAR L)
           RB)
        (CDR L))
      (T ERRL))))

(SFONTERM
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((OR (EQ (CAR L)
               AS)
           (EQ (CAR L)
               SL))
        (STERM (CDR L)))
      (T L))))

(SEONFACTOR
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((NULL (CDR L))
        L)
      ((AND (EQ (CAR L)
                AS)
            (EQ (CADR L)
                AS))
        (SFACTOR (CDDR L)))
      (T L))))

(SCONSTANT
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      (T (SEON (SPREFIX L))))))

(SPREFIX
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((DIGITP (CAR L))
        (SINDOTON (CDR L)))
      ((EQ (CAR L)
           PE)
        (SPREDOT (CDR L)))
```

```
        (T ERRL))))

(SPREDOT
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((DIGITP (CAR L))
        (SINTEGERON (CDR L)))
      (T ERRL))))

(SINTEGERON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((DIGITP (CAR L))
        (SINTEGERON (CDR L)))
      (T L))))

(SINDOTON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((DIGITP (CAR L))
        (SINDOTON (CDR L)))
      ((NOT (EQ (CAR L)
                PE))
        L)
      ((NULL (CDR L))
        NIL)
      ((DIGITP (CADR L))
        (SINTEGERON (CDDR L)))
      (T (CDR L)))))

(SEON
  (LAMBDA (L)
    (COND
      ((NULL L)
        L)
      ((EQ (CAR L)
           (QUOTE E))
        (SSINT (CDR L)))
      (T L))))

(SSINT
  (LAMBDA (L)
    (COND
      ((NULL L)
        ERRL)
      ((DIGITP (CAR L))
        (SINTEGERON (CDR L)))
      ((NULL (CDR L))
```

```
          ERRL)
         ((AND (SIGNP (CAR L))
               (DIGITP (CADR L)))
          (SINTEGERON (CDDR L)))
         (T ERRL))))

(SVAR
  (LAMBDA (S)
    (COND
      ((NULL S)
       ERRL)
      (T (SBEXPLON (SID S))))))

(SID
  (LAMBDA (S)
    (COND
      ((NULL S)
       ERRL)
      ((LETTERP (CAR S))
       (SANON (CDR S)))
      (T ERRL))))

(SANON
  (LAMBDA (S)
    (COND
      ((NULL S)
       S)
      ((ANP (CAR S))
       (SANON (CDR S)))
      (T S))))

(SBEXPLON
  (LAMBDA (S)
    (COND
      ((NULL S)
       S)
      ((EQ (CAR S)
           LB)
       (SBEXPL S))
      (T S))))

(SBIDLIST
  (LAMBDA (S)
    (COND
      ((NULL S)
       ERRL)
      ((EQ (CAR S)
           LB)
       (SRB (SIDLIST (CDR S))))
      (T ERRL))))

(SIDLIST
  (LAMBDA (S)
    (COND
```

```
        ((NULL S)
          ERRL)
        (T (SCOIDLISTON (SID S))))))

 (SCOIDLISTON
   (LAMBDA (S)
     (COND
       ((NULL S)
         S)
       ((EQ (CAR S)
            CO)
         (SIDLIST (CDR S)))
       (T S))))

 (SRINTLIST
   (LAMBDA (S)
     (COND
       ((NULL S)
         ERRL)
       ((EQ (CAR S)
            LB)
         (SRB (SINTLIST (CDR S))))
       (T ERRL))))

(SINTLIST
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      (T (SCOINTLISTON (SINTEGERL S))))))

(SCOINTLISTON
  (LAMBDA (S)
    (COND
      ((NULL S)
        S)
      ((EQ (CAR S)
           CO)
        (SINTLIST (CDR S)))
      (T S))))

(SINTEGERL
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((NOT (DIGITP (CAR S)))
        ERRL)
      ((EQUAL (CAR S)
           0)
        ERRL)
      (T (PROG (N R)
               (SETQ N (LENGTH S))
               (SETQ R (SINTEGER S))
```

```
                (SETQ N (DIFFERENCE N (LENGTH R)))
                (COND
                  ((GREATERP N 5)
                    (RETURN ERRL)))
                (RETURN R)
              )))))

(SINTEGER
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((DIGITP (CAR S))
        (SINTEGERON (CDR S)))
      (T ERRL))))

(SINTTRIPLE
  (LAMBDA (S)
    (SCOINTPAIR (SINTEGER S))))

(SCOINTPAIR
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((EQ (CAR S)
           CO)
        (SINTPAIR (CDR S)))
      (T ERRL))))

(SINTPAIR
  (LAMBDA (S)
    (SCOINT (SINTEGER S))))

(SCOINT
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((EQ (CAR S)
           CO)
        (SINTEGER (CDR S)))
      (T ERRL))))

(SDIMLIST
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      (T (SCODIMLISTON (SBLIMLIST (SID S)))))))

(SCODIMLISTON
  (LAMBDA (S)
    (COND
```

```
        ((NULL S)
          S)
        ((EQ (CAR S)
             CO)
          (SDIMLIST (CDR S)))
        (T S))))

(SBLIMLIST
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((EQ (CAR S)
           LB)
        (SRB (SLIMLIST (CDR S))))
      (T ERRL))))

(SLIMLIST
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      (T (SCOLIMLISTON (SLIM S))))))

(SCOLIMLISTON
  (LAMBDA (S)
    (COND
      ((NULL S)
        S)
      ((EQ (CAR S)
           CO)
        (SLIMLIST (CDR S)))
      (T S))))

(SLIM
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((EQ (CAR S)
           MI)
        (SSINT (SSL (SINTEGER (CDR S)))))
      (T (SSLSINTON (SSINT S))))))

(SSLSINTON
  (LAMBDA (S)
    (COND
      ((NULL S)
        S)
      ((EQ (CAR S)
           SL)
        (SSINT (CDR S)))
      (T S))))
```

```
(SSL
  (LAMBDA (S)
    (COND
      ((NULL S)
        ERRL)
      ((EQ (CAR S)
           SL)
        (CDR S))
      (T ERRL))))

(DYPROG
  (LAMBDA (P PRED)
    (PROG (U V W X)
          (SETQ U 0)
          (SETQ V (BUILD P))
     LOOP(SETQ X (PEELOFF U V W))
          (COND
            ((NULL X)
              (RETURN NIL)))
          (SETQ U (CADR X))
          (COND
            ((PRED (CAR X))
              (RETURN (CAR X))))
          (SETQ W (CADDR X))
          (GO LOOP)
    )))

(EXP
  (LAMBDA (P)
    (DEXP P)))

(DBINTLIST
  (LAMBDA (P)
    (DYPROG P (FUNCTION BINTLISTP))))

(DEXP
  (LAMBDA (P)
    (DYPROG P (FUNCTION EXPP))))

(VARIABLEP
  (LAMBDA (S)
    (NULL (SVAR S))))

(IDENTP
  (LAMBDA (S)
    (NULL (SID S))))

(BIDLISTP
  (LAMBDA (S)
    (NULL (SBIDLIST S))))

(IDLISTP
  (LAMBDA (S)
    (NULL (SIDLIST S))))
```

```
(BINTLISTP
  (LAMBDA (S)
    (NULL (SBINTLIST S))))

(INTLISTP
  (LAMBDA (S)
    (NULL (SINTLIST S))))

(INTEGERP
  (LAMBDA (S)
    (NULL (SINTEGER S))))

(SINTP
  (LAMBDA (S)
    (NULL (SSINT S))))

(INTTRIPLEP
  (LAMBDA (S)
    (NULL (SINTTRIPLE S))))

(INTPAIRP
  (LAMBDA (S)
    (NULL (SINTPAIR S))))

(DIMLISTP
  (LAMBDA (S)
    (NULL (SDIMLIST S))))

(LIMLISTP
  (LAMBDA (S)
    (NULL (SLIMLIST S))))

(BINTLIST
  (LAMBDA (P)
    (COND
      ((LESSP (LENGTH P)
              3)
        NIL)
      ((NOT (MEMBERLP LB (CAR P)))
        NIL)
      ((NOT (MEMBERLP RB (LAST1 P)))
        NIL)
      (T ((LAMBDA (X)
            (COND
              ((NULL X)
                NIL)
              (T (CONS LB (APPEND X (QUOTE (RB)))))))
          (INTLIST (INSIDE P)))))))

(LAST1
  (LAMBDA (P)
    (CAR (LAST P))))
```

```
(MEMBERLP
  (LAMBDA (X L)
    (COND
      ((NULL L)
        NIL)
      ((EQ X (CAAR L))
        T)
      (T (MEMBERLP X (CDR L))))))

(INSIDE
  (LAMBDA (P)
    (INSIDE1 (CDR P)
      NIL)))

(INSIDE1
  (LAMBDA (P O)
    (COND
      ((NULL (CDR P))
        (REVERSE O))
      (T (INSIDE1 (CDR P)
          (CONS (CAR P)
            O))))))

(INTLIST
  (LAMBDA (P)
    (COND
      ((NULL P)
        NIL)
      ((DIGCOPP P)
        (INTLIST1 (LINTLIST P NIL)))
      (T NIL))))

(DIGCOPP
  (LAMBDA (P)
    (COND
      ((NULL P)
        T)
      ((NOT (DIGCOLP (CAR P)))
        NIL)
      (T (DIGCOPP (CDR P))))))

(DIGCOLP
  (LAMBDA (L)
    (COND
      ((NULL L)
        NIL)
      ((DIGCOP (CAAR L))
        T)
      (T (DIGCOLP (CDR L))))))

(DIGCOP
  (LAMBDA (X)
    (COND
      ((OR (EQ X CO)
```

```
            (DIGITP X))
          T)
        (T NIL))))

(LINTLIST
  (LAMBDA (P Q)
    (COND
      ((NULL P)
        (REVERSE Q))
      (T (LINTLIST (CDR P)
          (CONS (DIGCO (CAR P))
            Q))))))

(DIGCO
  (LAMBDA (L)
    (COND
      ((DIGITP (CAAR L))
        (COMBINE (CAR L)
          (COON (CDR L))))
      ((EQ (CAAR L)
          CO)
        (COMBINE (CAR L)
          (DIGON (CDR L))))
      (T (DIGCO (CDR L))))))

(COMBINE
  (LAMBDA (X Y)
    (COND
      ((NULL Y)
        (LIST X))
      (T (LIST X Y)))))

(COON
  (LAMBDA (L)
    (COND
      ((NULL L)
        NIL)
      ((EQ (CAAR L)
          CO)
        (CAR L))
      (T (COON (CDR L))))))

(DIGON
  (LAMBDA (L)
    (COND
      ((NULL L)
        NIL)
      ((DIGITP (CAAR L))
        (CAR L))
      (T (DIGON (CDR L))))))

(INTLIST1
  (LAMBDA (P)
    (PROG (U V W)
```

```
            (SETQ U (PICKDIGIT (CAR P)))
            (COND
              ((NULL U)
                (RETURN NIL)))
            (COND
              ((NULL (CDR P))
                (RETURN (LIST U))))
            (SETQ W (PICKDIGIT (LAST1 P)))
            (COND
              ((NULL W)
                (RETURN NIL)))
            (COND
              ((LESSP (LENGTH P)
                   3)
                (RETURN (LIST U W))))
            (SETQ U (LIST (LIST (LIST U 100))))
            (SETQ W (LIST (LIST (LIST W 100))))
            (SETQ V (DINTLIST (APPEND U (APPEND (INSIDE P)
                      W))))
            (COND
              ((NULL V)
                (RETURN NIL)))
            (RETURN V)
        )))

(DINTLIST
  (LAMBDA (P)
    (DYPROG P (FUNCTION INTLISTP))))

(PICKDIGIT
  (LAMBDA (L)
    (COND
      ((NULL L)
        NIL)
      ((DIGITP (CAAR L))
        (CAAR L))
      (T (PICKDIGIT (CDR L))))))

(IDENT
  (LAMBDA (P)
    ((LAMBDA (A)
        (COND
          ((NULL A)
            NIL)
          ((GREATERP (LENGTH A)
               8)
            NIL)
          ((NULL (CDR A))
            (STARTVAR (CAR A)))
          (T (PROG (U V)
                   (SETQ U (STARTVAR (CAR A)))
                   (COND
                     ((NULL U)
                       (RETURN NIL)))
```

```
                    (SETQ V (RESTVAR (CDR A)
                        NIL))
                    (COND
                      ((NULL V)
                        (RETURN NIL)))
                    (RETURN (APPEND U V))
                  ))))
        (ALT P))))

(STARTVAR
  (LAMBDA (K)
    (COND
      ((NULL K)
        NIL)
      ((LETTERP (CAR K))
        (LIST (CAR K)))
      (T (STARTVAR (CDR K))))))

(RESTVAR
  (LAMBDA (A B)
    (COND
      ((NULL A)
        (REVERSE B))
      (T ((LAMBDA (Y)
            (COND
              ((NULL Y)
                NIL)
              (T (RESTVAR (CDR A)
                  (CONS Y B)))))
          (RESTVAR1 (CAR A)))))))

(RESTVAR1
  (LAMBDA (K)
    (COND
      ((NULL K)
        NIL)
      ((ANP (CAR K))
        (CAR K))
      (T (RESTVAR1 (CDR K))))))

(VAR
  (LAMBDA (P)
    (IDENT P)))
)
(PRINT (QUOTE F032A))
(SETQQ F032A (SPACE CLPSE NEL MATCH2 MATCH GM STYPE ST NUMVAL
ALT1 ALT NUM2 NUM3 NUM1 UGO1 NMTCH MEM2 UGO UGOTO UGOX ALLNUMP
HEAD HEAD1 TAIL NUMEM UPAUSE UASGT UCGT UCGT2 DESPACE UIFS UIFL
UIFS1 UIF UDO UDO1 FRONTDO BACKDO BACKDO1 BACKDO2 UAS UASSIGN
UASSIGN1 UFN UFN1 UCALL UCALL1 UCALL2 USBRTN USBRTN1 USBRTN2
UCOM UDIM UACC UPNCH UPNTP UREAD UTYPE UACCTP UPRNT GENIO UEF
URWND UBSP INIT INIT1 DECMOD2 DECMOD21 PMLAB SEG EXPH5 OP EXPH3
EXPH4 EXPH41 EXPH2 DBD EXPH EXPH1 LETTERP DIGITP SPECIALP ANP
SIGNP EXPP EXPLP SEXPL SCOEXPLON SEXP SUEXP SSEXPON STERM SFACTOR
```

```
SPRIMARY SVFON SBEXP SBEXPL SRB SFONTERM SEONFACTOR SCONSTANT
SPREFIX SPREDOT SINTEGERON SINDOTON SEON SSINT SVAR SID SANON
SBEXPLON SBIDLIST SIDLIST SCOIDLISTON SBINTLIST SINTLIST SCOINTLISTON
SINTEGERL SINTEGER SINTTRIPLE SCOINTPAIR SINTPAIR SCOINT SDIMLIST
SCODIMLISTON SBLIMLIST SLIMLIST SCOLIMLISTON SLIM SSLSINTON
SSL DYPROG EXP DBINTLIST DEXP VARIABLEP IDENTP BIDLISTP IDLISTP
BINTLISTP INTLISTP INTEGERP SINTP INTTRIPLEP INTPAIRP DIMLISTP
LIMLISTP BINTLIST LAST1 MEMBERLP INSIDE INSIDE1 INTLIST DIGCOPP
DIGCOLP DIGCOP LINTLIST DIGCO COMBINE COON DIGON INTLIST1 DINTLIST
PICKDIGIT IDENT STARTVAR RESTVAR RESTVAR1 VAR))
(PRINT (QUOTE VQ32A))
(SETQQ VQ32A (H HE H1 H2 H3 T O SP LB ES AS SL PL MI PE CO DS
RB ERRL LETTERLIST DIGITLIST SPECIALLIST))
(SETQQ H 40)
(SETQQ HE 6)
(SETQQ H1 30)
(SETQQ H2 30)
(SETQQ H3 40)
(SETQQ T T)
(SETQQ O O)
(SETQQ SP SP)
(SETQQ LB LB)
(SETQQ ES ES)
(SETQQ AS AS)
(SETQQ SL SL)
(SETQQ PL PL)
(SETQQ MI MI)
(SETQQ PE PE)
(SETQQ CO CO)
(SETQQ DS DS)
(SETQQ RB RB)
(SETQQ ERRL (ERR))
(SETQQ LETTERLIST (A B C D E F G H I J K L M N O P Q R S T U
V W X Y Z))
(SETQQ DIGITLIST (0 1 2 3 4 5 6 7 8 9))
(SETQQ SPECIALLIST (LB ES AS SL PL MI PE CO DS RB))
```

```
(PRIN1 (QUOTE THIS≤ FILE WAS CREATED ON ≤))
(PRINT (QUOTE 6-JUNE-1967))
(DEFINEQ

(BUILD
  (LAMBDA (ARR)
    ((LAMBDA (U)
        (CONS (ORDERED (CAR U))
          (CDR U)))
      (BUILD1 ARR (CAR ARR)
        NIL
        NIL
        NIL))))

(BUILD1
  (LAMBDA (ARR CARR ANS CANSW1 CANSW2)
    (COND
      ((NULL CARR)
        (COND
          ((NULL (CDR ARR))
            (CONS CANSW2 ANS))
          ((NULL ANS)
            (BUILD1 (CDR ARR)
              (CADR ARR)
              (LIST CANSW2)
              CANSW2
              NIL))
          (T (BUILD1 (CDR ARR)
              (CADR ARR)
              (CONS CANSW2 ANS)
              CANSW2
              NIL))))
      (T (BUILD1 ARR (CDR CARR)
          ANS
          CANSW1
          (ADDON (CAAR CARR)
            (CADAR CARR)
            CANSW1
            CANSW1
            CANSW2))))))

(ADDON
  (LAMBDA (I A B C D)
    (COND
      ((NULL B)
        (PUTIN A (CONS I 0)
          D))
      ((NULL C)
        D)
      (T (ADDON I A B (CDR C)
          (PUTIN (PLUS A (CAAR C))
            (CONS I (CAAR C))
            D)))))))
```

```
(PUTIN
  (LAMBDA (VAL PAIR OLD)
    (COND
      ((NULL OLD)
        (LIST (LIST VAL PAIR)))
      ((EQUAL (CAAR OLD)
          VAL)
        (CONS (APPEND (LIST VAL PAIR)
            (CDAR OLD))
          (CDR OLD)))
      (T (CONS (CAR OLD)
          (PUTIN VAL PAIR (CDR OLD)))))))

(ORDERED
  (LAMBDA (A)
    (COND
      ((NULL (CDR A))
        A)
      (T ((LAMBDA (U)
            (CONS U (ORDERED (DELETE U A))))
          (LARGEST (CAR A)
            (CDR A)))))))

(LARGEST
  (LAMBDA (C D)
    (COND
      ((NULL D)
        C)
      ((LESSP (CAR C)
          (CAAR D))
        (LARGEST (CAR D)
          (CDR D)))
      (T (LARGEST C (CDR D))))))

(DELETE
  (LAMBDA (ELEM LISTA)
    (COND
      ((NULL LISTA)
        NIL)
      ((EQUAL ELEM (CAR LISTA))
        (CDR LISTA))
      (T (CONS (CAR LISTA)
          (DELETE ELEM (CDR LISTA)))))))

(PEELOFF
  (LAMBDA (OLDVAL ANS OTHERS)
    (COND
      ((NULL (CAR ANS))
        NIL)
      ((ZEROP OLDVAL)
        (DOWN1 (NXT ANS)
          (CAAAR ANS)
          (CDR ANS)
          2
```

```
            (FIX0 OTHERS ANS 1 (CAAAR ANS))
            (LIST (CHO ANS))))
        ((GREATERP (CAAAR ANS)
            OLDVAL)
          (PEELOFF OLDVAL (CONS (CDAR ANS)
              (CDR ANS))
            OTHERS))
        ((NULL OTHERS)
          (PEELOFF 0 (CONS (CDAR ANS)
              (CDR ANS))
            OTHERS))
        ((NOT (BELONG 1 OTHERS))
          (DOWN1 (NXT ANS)
            OLDVAL
            (CDR ANS)
            2
            (FIX0 OTHERS ANS 1 (CAAAR ANS))
            (LIST (CHO ANS))))
        (T (CHOOSE1 OLDVAL 1 (CDAAR ANS))))))

(CHOOSE1
  (LAMBDA (NEWVAL CTR CHOICES)
    (COND
      ((EQUAL (CAADAR OTHERS)
          1)
        (COND
          ((EQUAL (CDADAR OTHERS)
              CTR)
            (COND
              ((NULL (CDR CHOICES))
                (PEELOFF 0 (CONS (CDAR ANS)
                    (CDR ANS))
                  NIL))
              (T (DOWN1 (CDADR CHOICES)
                  NEWVAL
                  (CDR ANS)
                  2
                  (FIX1 CTR 1)
                  (LIST (CAADR CHOICES))))))
          (T (CHOOSE1 NEWVAL (ADD1 CTR)
              (CDR CHOICES)))))
      (T (DOWN1 (CDR (FIND 1 CHOICES OTHERS 1))
          NEWVAL
          (CDR ANS)
          2
          OTHERS
          (LIST (CAR (FIND 1 CHOICES OTHERS 1))))))))

(CHOOSE
  (LAMBDA (LVL CTR CHOICE)
    (COND
      ((EQUAL (CAADAR OTHERS)
          LVL)
        (COND
```

```
            ((EQUAL (CDADAR OTHERS)
                CTR)
              (COND
                ((NULL (CDR CHOICE))
                  (BACKUP LVL OTHERS ANS1 (REVERSE NLIST)
                    NIL))
                (T (DOWN1 (CDADR CHOICE)
                    NEWVAL
                    (CDR ANS1)
                    (ADD1 LVL)
                    (FIX1 CTR LVL)
                    (CONS (CAADR CHOICE)
                      NLIST)))))
            (T (CHOOSE LVL (ADD1 CTR)
                (CDR CHOICE)))))
        (T (DOWN1 (CDR (FIND LVL CHOICE OTHERS 1))
            NEWVAL
            (CDR ANS1)
            (ADD1 LVL)
            OTHERS
            (CONS (CAR (FIND LVL CHOICE OTHERS 1))
              NLIST))))))

(FIX1
  (LAMBDA (A B)
    (CONS (LIST (CAAR OTHERS)
            (CONS B (ADD1 A)))
      (CDR OTHERS))))

(FIX0
  (LAMBDA (OTHERS A B C)
    (COND
      ((NULL (CDDAAR A))
        OTHERS)
      (T (CONS (LIST C (CONS B 1))
          OTHERS)))))

(FIND
  (LAMBDA (LVL A B C)
    (COND
      ((EQUAL LVL (CAADAR B))
        (COND
          ((EQUAL C (CDADAR B))
            (CAR A))
          (T (FIND LVL (CDR A)
              B
              (ADD1 C))))))
      (T (FIND LVL A (CDR B)
          C)))))

(CHO
  (LAMBDA (X)
    (CAAR (CDAAR X))))
```

```
(NXT
  (LAMBDA (X)
    (CDAR (CDAAR X))))

(DOWN1
  (LAMBDA (LEVEL NEWVAL ANS1 I OTHERS NLIST)
    (COND
      ((ZEROP LEVEL)
        (LIST NLIST NEWVAL OTHERS))
      ((NOT (EQUAL (CAAAR ANS1)
                   LEVEL))
        (DOWN1 LEVEL NEWVAL (CONS (CDAR ANS1)
                                  (CDR ANS1))
          I
          OTHERS
          NLIST))
      ((NOT (BELONG I OTHERS))
        (DOWN1 (NXT ANS1)
          NEWVAL
          (CDR ANS1)
          (ADD1 I)
          (FIXO OTHERS ANS1 I LEVEL)
          (CONS (CHO ANS1)
            NLIST)))
      (T (CHOOSE I 1 (CDAAR ANS1))))))

(BACKUP
  (LAMBDA (COUNT OTHERS2 ANSW2 NLISTB NLISTA)
    (COND
      ((OR (NULL OTHERS2)
           (NULL ANSW2))
        (COND
          ((CDAR ANS)
            (PEELOFF 0 (CONS (CDAR ANS)
                             (CDR ANS))
              NIL))
          (T NIL)))
      ((EQUAL COUNT (CAADAR OTHERS2))
        (COND
          ((EQUAL (CAAR OTHERS2)
                  (CAAAR ANSW2))
            (RIGHTSPOT (FINDNXT (CDAAR ANSW2)
                                1
                                OTHERS2)))
          (T (BACKUP COUNT OTHERS2 (CONS (CDAR ANSW2)
                                         (CDR ANSW2))
                     NLISTB
                     NLISTA))))
      ((NULL NLISTB)
        (BACKUP (ADD1 COUNT)
          OTHERS2
          (CDR ANSW2)
          NIL
          NLISTA)))
```

```
      (T (BACKUP (ADD1 COUNT)
                 OTHERS2
                 (CDR ANSW2)
                 (CDR NLISTB)
                 (CONS (CAR NLISTB)
                       NLISTA))))))

(RIGHTSPOT
  (LAMBDA (U)
    (COND
      (U (DOWN1 (CDR U)
                NEWVAL
                (CDR ANSW2)
                (ADD1 COUNT)
                (CONS (LIST (CAAR OTHERS2)
                            (CONS COUNT (ADD1 (CDADAR OTHERS2))))
                      (CDR OTHERS2))
                (CONS (CAR U)
                      (CDR NLISTA))))
      ((AND NLISTA (CDR NLISTA))
        (BACKUP 1 (CDR OTHERS2)
                ANS
                (CDR (REVERSE (CDR NLISTA)))
                (LIST (CAR (REVERSE NLISTA)))))
      (NLISTA (BACKUP 1 (CDR OTHERS2)
                ANS
                NIL
                NIL))
      (NLISTB (BACKUP 1 (CDR OTHERS2)
                ANS
                (CDR NLISTB)
                (LIST (CAR NLISTB))))
      (T NIL))))

(FINDNXT
  (LAMBDA (LISTE CTR OTHS)
    (COND
      ((NULL LISTE)
        NIL)
      ((EQUAL CTR (CDADAR OTHS))
        (COND
          ((NULL (CDR LISTE))
            NIL)
          (T (CADR LISTE))))
      ((NULL (CDR LISTE))
        NIL)
      (T (FINDNXT (CDR LISTE)
                  (ADD1 CTR)
                  OTHS)))))

(BELONG
  (LAMBDA (EL LISTE)
    (COND
      ((NULL LISTE)
```

```
        NIL)
       ((EQUAL EL (CAADAR LISTE))
        T)
       ((NULL (CDR LISTE))
        NIL)
       (T (BELONG EL (CDR LISTE))))))

(STE2
  (LAMBDA NIL
    (ST E2)))

(STE3
  (LAMBDA NIL
    (ST E3)))

(STE4
  (LAMBDA NIL
    (ST E4)))

(STE5
  (LAMBDA NIL
    (ST E5)))

(STE6
  (LAMBDA NIL
    (ST E6)))

(STED0
  (LAMBDA NIL
    (ST ED0)))

(STED1
  (LAMBDA NIL
    (ST ED1)))

(STED2
  (LAMBDA NIL
    (ST ED2)))

(STED3
  (LAMBDA NIL
    (ST ED3)))
)
(PRINT (QUOTE F10328))
(SETQQ F10328 (BUILD BUILD1 ADDON PUTIN ORDERED LARGEST DELETE
PEELOFF CHOOSE1 CHOOSE FIX1 FIX0 FIND CH0 NXT DOWN1 BACKUP RIGHTSPOT
FINDNXT BELONG STE2 STE3 STE4 STE5 STE6 STED0 STED1 STED2 STED3))
```

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP N/A

3. REPORT TITLE

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Report No. 27, Fifth Quarterly Report - 1 February 1967 to 31 May 1967

5. AUTHOR(S) (First name, middle initial, last name)

Richard O. Duda John H. Munson

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1967	106	1

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DA 28-043 AMC-01901(E) b. PROJECT NO.	SRI Project 5864 Fifth Quarterly Report
c. d.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) ECOM-01901-27

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	U.S. Army Electronics Command Fort Monmouth, New Jersey 07703

13. ABSTRACT

This report describes the continuing development of preprocessing, character-classification, and context-analysis techniques for hand-printed text, such as computer coding sheets in the FORTRAN language.

We discuss preprocessing operations developed to find topological features such as the perimeter, the convex hull, concavities, enclosures, and spurs of a black/white quantized figure. We describe the present status of a program, TOPO 2, which classifies characters on the basis of these and other topological features. Finally, we detail the component functions of the LISP program developed to perform FORTRAN syntax analysis on a statement-by-statement basis.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Pattern Recognition						
Adaptive Systems						
Learning Machines						
Preprocessing						
Classification						
Context Analysis						
FORTRAN Syntax Analysis						