RECOGNITION OF PRINTED BANGLA CHARACTERS
by
SYNTACTIC METHOD OF PATTERN RECOGNITION

By
MD. ABUL KASHEM MIA

A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, BUET, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ENGINEERING

বাংলাদেশ প্রকৌশল বিশ্ববিদ্যালয়
নং 85057
তাং 7-10-92
লাইব্রেরী, ঢাকা

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
AUGUST, 1992

#85057#

623.819598
1992
ABU

RECOGNITION OF PRINTED BANGLA CHARACTERS
by
SYNTACTIC METHOD OF PATTERN RECOGNITION

A thesis submitted by

MD. ABUL KASHEM MIA
Roll No. 891818P, Registration No. 83120,
for the partial fulfillment of the degree of
M.Sc. Engg. in Computer Science & Engineering.
Examination held on: August 02, 1992.

Approved as to style and contents by:

2/8/92

DR. SYED MAHBUBUR RAHMAN — Chairman and Supervisor
Head,
Department of Computer Science & Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

2/8/92

DR. MOHAMMAD KAYKOBAD — Member
Assistant Professor,
Department of Computer Science & Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

MR. MD. ABDUS SATTAR — Member
Assistant Professor,
Department of Computer Science & Engineering
B.U.E.T., Dhaka-1000, Bangladesh.

2/8/92

DR. MD. LUTFAR RAHMAN — Member (External)
Associate Professor,
Department of Applied Physics & Electronics
University of Dhaka, Dhaka-1000, Bangladesh.

CERTIFICATE

This is to certify that the work presented in this thesis paper is the outcome of the investigation carried out by the candidate under the supervision of Dr. Syed Mahbubur Rahman in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. It is also declared that neither of this thesis nor any part thereof has been submitted or is being concurrently submitted anywhere else for the award of any degree or diploma or for publication.

SMRahman 2/8/92

Signature of the Supervisor

2/8/92

Signature of the Author

ACKNOWLEDGEMENT

The author feels pleased to have the opportunity of expressing his heart-felt and most sincere gratitude to a number of people who have helped him in course of this research work.

First and foremost, the author wishes to acknowledge his profound indebtness to Dr. Syed Mahbubur Rahman, Professor and Head, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology. Keen interest of Dr. Rahman in the field of Application of Computer Facilities in Bangla has influenced the author to carry out a research work in the field of optical recognition of Bangla characters. His constant supervision, constructive criticism, valuable advice, scholarly guidance and encouragement at all stages of this research have made it possible to complete this research.

The author expresses his utmost gratitude to Dr. Mohammad Kaykobad, Assistant Professor, Department of Computer Science and Engineering, BUET, for the valuable discussions occasionally made with him.

The author acknowledges with sincere thanks the all-out co-operation and services rendered by the faculty members and staff of the CSE Department specially by Mr. Syed Ahsanul Karim.

Finally, to all who contributed in any way, the author desires to express his deep appreciations and sincere feelings.

ABSTRACT

A syntactic pattern recognition scheme suitable for recognizing printed Bangla alphabetic characters irrespective of their size up to a certain limit has been developed.

The scheme operates on a representation of one character at a time. The character matrix having value '0' or '1' is obtained by means of a scanner. The strokes are obtained from '0' to '1' transitions of the matrix. Some filtering techniques are used to clean the noise present in the strokes. From the relationship among the strokes representing the structure of a character, numeric codes are generated. A dictionary of codes is searched to find a matching code corresponding to the code generated from the input character. Codes acquired from five sets of test characters having variable size are collected and formatted to construct the dictionary. Most of the characters have got more than one code due to the printing styles, sizes and quality of the digitized output. There are several groups of characters which show same code due to structural similarity among them. These characters are distinguished by comparing the stroke ratio of a specific stroke for different members of the same group.

A character recognition software in C language has been developed to implement the proposed scheme in IBM PC or compatibles. The scheme has been successfully applied to different sized characters with less than 6 errors per 100 processed characters.

CONTENT

TITLE OF THE THESIS i

CERTIFICATE OF APPROVAL ii

CERTIFICATE OF RESEARCH iii

ACKNOWLEDGEMENT iv

ABSTRACT v

CONTENT vi

CHAPTER 1. INTRODUCTION

1.1 Introduction to Pattern Recognition 1

1.2 Pattern Recognition Areas 3

1.3 Necessity of Character Recognition by Machines 4

1.4 Elements of a Pattern Recognition System 4

1.5 Approaches to Pattern Recognition 7

1.5.1 Decision-Theoretic Approach 7

1.5.2 Syntactic Approach 11

1.6 Remarks on Syntactic Approach vs. Decision-theoretic Approach 13

1.7 Literature Survey 15

1.8 Scope of the Present Work 19

CHAPTER 2. SYNTACTIC METHOD

2.1 Introduction 21

2.2 Syntactic Pattern Recognition System 25

2.3 Preprocessing Techniques 28

2.3.1 Encoding and Approximation 28

2.3.2 Filtering, Restoration and Enhancement 29

2.4 Picture Segmentation 29

2.4.1 Characteristics Feature Thresholding and Clustering 31

2.4.1.1 Thresholding 31

2.4.1.2 Clustering 33

2.4.2 Edge Detection 33

2.4.2.1 Parallel Techniques 34

2.4.2.2 Sequential Techniques 34

2.4.3 Region Extraction 35

2.4.3.1 Region Merging 35

2.4.3.2 Region Dividing 36

2.4.3.3 Region Merging and Dividing 36

2.5 Selection of Pattern Primitives 36

2.6 Primitive Recognition 39

2.7 Pattern Grammar 40

2.8 Recognition Procedure 42

CHAPTER 3. DEVELOPMENT OF BANGLA CHARACTER RECOGNITION SYSTEM

3.1 Introduction 45

3.2 Methodology of the Recognition Process 45

3.3 Input: Digitization 47

3.3.1 Scanner 48

3.4 Preprocessor 49

3.4.1 Get Letter 51

3.4.2 Get Letter Contour: Contour Tracing 53

3.4.3 Filtering 56

3.5 Extraction of Strokes (Morphs) 58

3.6 Encoding of Strokes (Morphs) 61

3.6.1 Selection of Strokes 61

3.6.2 Finding Stroke Code 61

3.7 Encoding of Characters 66

3.8 Recognition of Characters 67

3.8.1 Characters Having More Than One Code 67

3.8.2 Characters With Same Code 77

3.8.3 Development of a Dictionary 84

CHAPTER 4. DISCUSSIONS, CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

4.1 General Remarks 85

4.2 Results 86

4.3 Conclusions 88

4.4 Suggestion for Future Work 90

REFERENCES 92

APPENDIX A. REGRESSION ANALYSIS FOR FINDING THE CO-EFFICIENTS OF A SECOND ORDER EQUATION 96

APPENDIX B. DICTIONARY SHOWING NUMERIC CODES CORRESPONDING TO ALL BANGLA CHARACTERS 98

APPENDIX C. MAIN PROGRAM FOR BANGLA CHARACTER RECOGNITION 104

APPENDIX D. EXTERNAL FUNCTION RecognizeChar() 131

CHAPTER ONE

INTRODUCTION

One of the most interesting aspects of the world is that it can be considered to be made up of patterns. A pattern is essentially an arrangement. It is characterized by the order of elements of which it is made, rather than by the intrinsic nature of these elements.

Norbert Wiener

CONTENT

CHAPTER 1. INTRODUCTION

1.1 Introduction to Pattern Recognition 1

1.2 Pattern Recognition Areas 3

1.3 Necessity of Character Recognition by Machines 4

1.4 Elements of a Pattern Recognition System 4

1.5 Approaches to Pattern Recognition 7

1.5.1 Decision-Theoretic Approach 7

1.5.2 Syntactic Approach 11

1.6 Remarks on Syntactic Approach vs. Decision-theoretic Approach 13

1.7 Literature Survey 15

1.8 Scope of the Present Work 19

বাংলাদেশ প্রকৌশল বিশ্ববিদ্যালয়
নং 85057
তাং 7-10-92
লাইব্রেরী, ঢাকা

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PATTERN RECOGNITION.

A *pattern* is a quantitative or structural description of an object or some other entity of interest.

Recognition is regarded as a basic attribute of human beings and other living organisms. According to the nature of patterns to be recognized, we may divide our acts of recognition into two major types: the recognition of concrete items and the recognition of abstract items. We recognize characters, pictures, music and objects around us. This process may be referred to as *sensory recognition*, which includes visual and aural pattern recognition. On the other hand, we can recognize an old argument or a solution to a problem without resorting to external stimuli. This process involves the recognition of abstract items and can be termed *conceptual recognition*.

Pattern recognition can be defined as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail [1]. The set of patterns to be recognized

could be a set of physical objects or a set of mental states. A *pattern class* is a set of patterns that share some common properties. The number of pattern classes is often determined by the particular application in mind. For example, consider the problem of Bangla alphabet recognition: we should have a problem of 51 classes, one for each of the 50 letters and a rejection class for all unacceptable characters. On the other hand, if we are interested in discriminating English characters from Bangla characters, we have only a three-class problem. In some problems, the exact number of classes may not be known initially, and it may have to be determined from the observations of many representative patterns. In this case, we would like to detect the possibility of having new classes of patterns as we observe more and more patterns. Human beings perform the task of pattern recognition in almost every instant of their working lives. Recently, scientists and engineers started to use machines for pattern recognition. The subject matter of *pattern recognition by machine* deals with techniques for assigning patterns to their respective classes, automatically and with as little human intervention as possible. In the case of alphabetic Bangla character recognition, the function of the character recognition machine is to identify each input character as being a member of one of the available pattern classes.

The study of pattern recognition problems may be logically divided into two major categories [1]:

1. The study of the pattern recognition capability of human beings and other living organisms.

2. The development of underlying theory and practical techniques for machine implementation of a given recognition task.

The first subject area falls in the domain of such disciplines as psychology, physiology, and biology. The second area is in the domain of engineering, computer science and applied mathematics.

1.2 PATTERN RECOGNITION AREAS.

Pattern recognition is a major area of activity that encompasses the processing of pictorial information obtained from interaction between science and society. Pattern recognition is needed for to communicate with the computing machines in human beings natural mode of communication (the human voice and written script) [2]. Pattern recognition is also concerned with the area of designing and making automata that can hear and understand what human being say and write, the automata that can speak and make people understand and the automata that can process pictorial information for human use with more and more efficiency. The research in pattern recognition encompasses the field of communication and computer science, mathematics and statistics, acoustics, phonetics, linguistics and psycholinguiltics, speech pathology, hematology, neoro-physiology, remote sensing techniques and similar other many aspects.

Research and development on pattern recognition and applications have been carried out in the following major groups:

i) Man-Machine communication.

ii) Bio-medical applications and diagnosis of pathological conditions by analyzing

X-ray and/or cytological slides.

iii) Natural resources estimation and planning in agriculture, forestry, hydrology, geological environment etc.

iv) Scientific and military applications.

v) Detection of crime and criminals.

In this thesis paper we are concerned with visual sensory recognition specifically in the field of alphabetic Bangla characters to be mainly used in the Man-Machine communication area.

1.3 NECESSITY OF CHARACTER RECOGNITION BY MACHINES.

Modern high speed computers can execute millions of instructions per seconds, but their capacity as information processing devices is limited by the type of input data that they can accept. Much of the information that we might like the computers to process is in pictorial form; this includes alphanumeric characters (printed text, hand writing etc.). Alphanumeric data can be manually converted into computer readable form, but much faster input rates can be achieved if scanning and optical recognition techniques are used.

Some applications of character recognition is shown in Table 1.1 [3].

1.4 ELEMENTS OF A PATTERN RECOGNITION SYSTEM.

The principal function of a pattern recognition system is to arrive at decisions concerning the class membership of the patterns with which it is confronted. Several major information translation processes take place between the time a pattern is input and a decision is made by the system. These processes, which are summarized

TABLE 1.1

SOME APPLICATIONS OF CHARACTER RECOGNITION

Problem	Input to recognition system	Output of recognition system
Bank checks.	Magnetic response wave form, Optical scanned image.	Numeric characters, Special symbols.
Automatic process of documents -- utility bills, credit card charges, sales and inventory documents.	Optical scanned image.	Alphanumeric characters, Special symbols.
Journal tape reading.	Optical scanned image.	Alphanumeric characters, Special symbols.
Page readers -- automatic type setting, input to computers, reading for the blind.	Optical scanned image.	Alphanumeric characters, Special symbols.
Label readers.	Optical scanned image.	Alphanumeric characters, Special symbols.
Address readers.	Optical scanned image.	Alphanumeric characters combined into postal codes, street names and city.
Other readers -- license plate readers, telephone traffic counter reader etc.	Optical scanned image.	Alphanumeric characters, Special symbols.

in block diagram form in Fig 1.1, extract from the input data the discriminatory information required for classification. The functions of the blocks shown in Fig 1.1 are described briefly as follows.

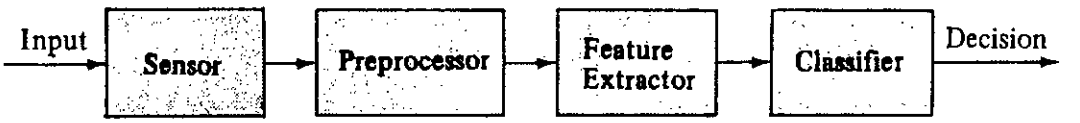


Fig. 1.1 Components of pattern recognition system.

The sensor is simply the measurement device that transforms the input patterns into a form suitable for machine manipulation. Although some simple pattern recognition systems operate on the input data directly from the sensor, it is common practice to follow the sensor with a preprocessor and feature extractor. The preprocessor removes unnecessary or corrupting elements from the measured data, while the feature extractor computes from the preprocessed data the features required for classification. Finally, these features are input into the classifier, whose function is to yield a decision concerning the class membership of the pattern being processed.

1.5 APPROACHES TO PATTERN RECOGNITION.

Approaches to pattern recognition system design may be divided into two principal categories [4]:

a) The decision-theoretic approach and

b) The syntactic approach.

1.5.1 Decision-Theoretic Approach.

In the decision-theoretic approach, a set of selected characteristics measurements, called 'features', are extracted from the patterns [1]. These features are supposed to be invariant or less sensitive with respect to the commonly encountered variations and also containing less redundancies. Under this proposition, pattern recognition can be considered as consisting of two subproblems. The first subproblem is what measurements should be taken from the input patterns. Usually, the decision of what to measure is rather subjective and also dependent on the practical situations (for example, the availability of measurements, the cost of measurements, etc.). The criterion of feature selection or ordering is often based on either the importance of the features in characterizing the patterns or the contribution of the features to the performance of recognition (i.e., the accuracy of recognition).

The second subproblem in pattern recognition is the problem of classification (or making a decision on the class assignment to the input patterns) based on the measurements taken from the selected features. The device or machine which extracts the feature measurements from input patterns is called a *feature extractor*. The device or machine which performs the function of classification is called a

classifier. A simplified block diagram of a pattern recognition system using decision theoretic approach is shown in Fig. 1.2.

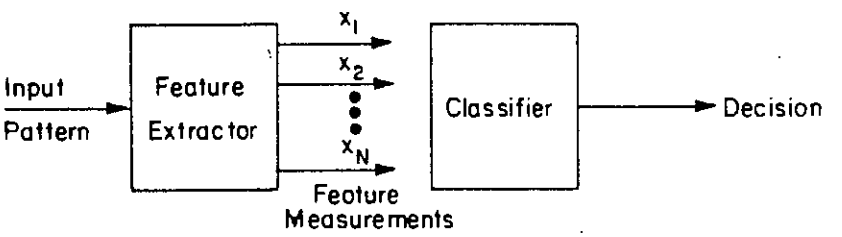


Fig. 1.2 Block diagram of a pattern recognition system using decision-theoretic approach.

It is often convenient to arrange the sensed data in the form of a *pattern vector*

$$\mathbf{x} = [x_1 \quad x_2 \quad \ldots \quad x_n]$$

where n is the number of measurements, and each component x_i of vector $\mathbf{x}$ assumes the value 1 or 0, depending on the state of the ith cell in the grid for a particular input.

The decision-theoretic approach is based on the utilization of decision functions for classifying pattern vectors. As an introduction to this concept, let us consider Fig. 1.3, which shows two pattern classes denoted by ω_1 and ω_2.

It is seen by inspection that these two classes can be separated by a straight line. Let

$$d(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 = 0.$$

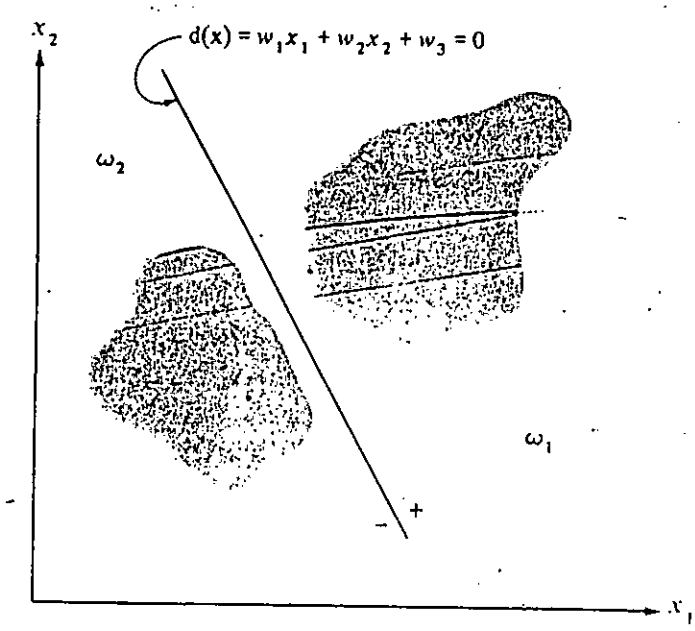


Fig. 1.3 A simple decision function for two pattern classes.

be the equation of a separating line where the w's are parameters, x_1 and x_2 are the pattern coordinate variables and $d(\mathbf{x})$ is used to denote $d(x_1, x_2)$. Any observation $\mathbf{x}$ that falls on the boundary yields $d(\mathbf{x}) = 0$ when substituted into the foregoing equation. Any pattern $\mathbf{x}$ from class ω_1 will yield a positive quantity when substituted into $d(\mathbf{x})$. In other words, all the patterns of this class lie on the positive side of the separating boundary. Similarly the patterns of class ω_2 lie on the negative side of the boundary. From these considerations, it is noted that $d(\mathbf{x})$ can be used as a *decision* (or *discriminant*) function, since, given a pattern $\mathbf{x}$ of unknown classification, we say that $\mathbf{x}$ belongs to class ω_1 if $d(\mathbf{x}) > 0$ or to class ω_2 if $d(\mathbf{x}) < 0$.

When there are more than two classes, one approach is to establish M decision functions $d_1(\mathbf{x}), d_2(\mathbf{x}), \ldots, d_M(\mathbf{x})$ with the property that if a pattern $\mathbf{x}$ belongs to class ω_i then

$$d_i(\mathbf{x}) > d_j(\mathbf{x}), \qquad j = 1, 2, \quad \ldots, M, \quad j \neq i,$$

where M is the number of classes. This relationship specifies a decision rule in which each pattern to be classified is substituted into all decision functions. The pattern is then assigned to the class whose decision yields the largest numerical value. Ties are resolved arbitrarily. It is noted that the simple example introduced above is a special case of this rule. Thus, if $M = 2$, we have that for all patterns of class ω_1, $d_1(\mathbf{x}) > d_2(\mathbf{x})$ and, conversely, for all patterns of class ω_2, $d_1(\mathbf{x}) < d_2(\mathbf{x})$. Defining $d(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x})$ leads to the condition $d(\mathbf{x}) > 0$ for $\mathbf{x}$ in ω_1 and $d(\mathbf{x}) < 0$ for $\mathbf{x}$ in ω_2. In general, the equation of the decision boundary separating classes ω_i and ω_j is given by

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0.$$

The foregoing concepts need not be restricted to two-dimensional patterns and linear decision boundaries. In the n-dimensional case, the decision functions may be expressed in the general form

$$d_k(\mathbf{x}) = \sum_{l=1}^{k} w_{kl}\phi_l(\mathbf{x}), \qquad k = 1, 2, \ldots, M$$

where the $\{\phi_l(\mathbf{x})\}$ are real, single-valued functions of the pattern $\mathbf{x}$ and $\{w_{kl}\}$ are the coefficients of the decision function corresponding to class ω_k. The usual approach is to specify first the functions $\{\phi_l(\mathbf{x})\}$. The problem then becomes one

of determining the coefficients $\{w_{kl}\}$ for each class, so that the foregoing decision rule will hold for as many patterns as possible. The decision-theoretic approach to pattern recognition deals with algorithms for estimating these parameters using sample patterns in a training process. Thus, if the pattern recognition system performs well during training with a well-chosen set of representative patterns, it may be expected to perform satisfactory when confronted with 'field' data during normal operation.

There exists numerous adaptive algorithms that can be used for training a pattern recognition system. These algorithms can be divided into (1) deterministic and (2) statistical procedures [1]. Deterministic algorithms deal with the estimation of the decision function coefficients directly from the patterns without resorting to statistical considerations, while statistical algorithms are based on the statistical properties of the various pattern populations under consideration. Although a study of these algorithms is outside the scope of this research work, it is important to bear in mind that the decision-theoretic approach to pattern recognition plays a very significant role in applied as well as theoretical work in this field.

1.5.2 Syntactic Approach.

The decision-theoretic approach discussed in the previous section is ideally suited for applications where patterns can be meaningfully represented in vector form. There are applications, however, where the structure of a pattern plays an important role in the classification process. In these situations, the decision-theoretic approach has serious drawbacks because it lacks a suitable formalism for

handling pattern structures and their relationships. For example, the decision-theoretic approach finds few applications to scene analysis, since in this case the structure and relationships of the various components of a scene are of fundamental importance in establishing a meaningful recognition scheme.

The syntactic approach to pattern recognition has been receiving increased attention during the past few years because it possesses the structure-handling capability lacked by the decision-theoretic approach. Syntactic pattern recognition is based on concepts from formal language theory, the origins of which may be traced to the middle of 1950s with the development of mathematical models of grammars by Noam Chomsky [1956]. One of the original goals of linguists working in this area was to develop mathematical models capable of describing natural languages such as English. The hope was that, if this could be done, it would be a relatively simple matter to 'teach' computers to interpret natural languages for the purposes of translation and problem solving.

Basic to the syntactic pattern recognition approach is the decomposition of patterns into sub-patterns of primitives, called *morphs*. To represent a pattern in terms of its subpatterns, we must segment or decompose the pattern and, in the meantime, identify (or recognize) the primitives and the relations in it. At this point, each pattern is represented by a set of primitives with specified syntactic operations. For example, in terms of the concatenation operation, each pattern is represented by a string of (concatenated) primitives, called *sentence*. The decision as to whether or not the representation (pattern) is syntactically correct (i.e. belongs to the class of patterns described by the given syntax or grammar) will be made by

the syntax analyzer or parser. A pattern is uniquely assigned to the ith class if it is a sentence of only language i, and no other language. A unique decision cannot be made if the sentence belongs to more than one language, and if a pattern is not a sentence of any of the languages under consideration, it is assigned to a rejection class.

1.6 REMARKS ON SYNTACTIC APPROACH VS. DECISION- THEORETIC APPROACH.

Some successful results have been obtained in applying syntactic approaches to character recognition, chromosome analysis, bubble chamber picture identification. However, the dichotomy of syntactic and decision-theoretic approaches appears to be convenient only from the view point of theoretical study. Many preprocessing and segmentation techniques are useful in both approaches, apparently because the selection of a particular preprocessing or segmentation technique now depends more on the type of the patterns under study than the recognition approach being applied.

The basic difference between the two approaches is that in decision-theoretic approach the 'features' are a set of numerical measurements taken from the pattern whereas the 'morphs' in syntactic model are subpatterns themselves. Statistical decision theory focuses entirely on statistical relationships among scalar features ignoring any other structural properties that characterized the pattern i.e. decision-theoretic approach only classifies the pattern whereas syntactic approach serves classification as well as description of the pattern.

However, with respect to practical pattern recognition problems, the syntactic and decision-theoretic approaches in many cases complement each other. When explicit structural information about the patterns is not considered important and the problem is primarily one of classification rather than of classification and description, there appears to be no real need for using syntactic methods. When, on the other hand, patterns are very rich in structural information and the recognition problem requires classification and description, the syntactic approach seems to be necessary. In many pattern recognition applications, the problem essentially falls between these two extreme cases. In other words, some sort of structural information about patterns is important and should be utilized, but simple and abstract pattern primitives may not be easy to extract, particularly for noisy and distorted patterns. Since in general, pattern primitives are defined more or less in terms of local properties of a pattern and contain little syntactic information, they probably can be more effectively recognized by decision-theoretic techniques. After the pattern primitives are extracted, a syntactic method would be applied for the recognition of the whole pattern. An appropriate utilization of the combination of these approaches may result in an efficient, practical system.

When patterns belonging to two or more different classes appear to have the same noisy version, it is accepted by two or more different pattern grammars. This situation results in the so called *ambiguity* in formal language theory. This ambiguity in structural descriptions cannot be discriminated by normal syntax analysis procedures. In such circumstances, the discrimination among different structural descriptions may be carried out by decision-theoretic techniques from the statistical

information about the pattern noise and distortion.

1.7 LITERATURE SURVEY.

Over the years, the field of pattern recognition has attracted workers from a variety of areas such as engineering, system theory, statistics, linguistics, psychology etc., resulting in a vast literature containing abstract mathematical approaches as well as highly pragmatic techniques. The literatures are scattered in a large number of journals in several fields. Some of the important literatures, relevant to the work, are discussed in this section.

L.N. Kanal in his paper [5] selectively surveyed contributions to major topics in pattern recognition during the period 1968 to 1974. This paper also provides a very useful bibliography on representative books and surveys on pattern recognition published during the above mentioned period. Theoretical models for automatic pattern recognition are contrasted with practical design methodology. He selectively discussed the research contributions to statistical and structural pattern recognition including contributions to error estimation and the experimental design of pattern classifiers. His paper concludes with a representative set of applications of pattern recognition technology.

In his paper [6] Kendall Preston Jr. gave a thorough comparison of analog and digital techniques used for pattern recognition. This paper reviews three major categories (electronic, acoustical and optical) of analog technology used in pattern recognition and predicts the future trends upon the analysis of performance advances which have taken place in both digital and analog fields during the past

decade.

Unger [7] observed that for any alphabet there must exist at least one finite set of characteristics that can be used to distinguish among the members. He suggested that there must exist a set of yes or no questions such that if these questions are answered with respect to any given figure then there will be only one member of the alphabet to which this figure can belong.

Horwitz and Shelton [8] described a class of techniques for character recognition. These techniques are characterized by the property that the only parameters of the input which are used are those which are independent of the position of the characters, i.e. these techniques are registration invariant.

Perotto [9] described a method, called 'morphotopological method for character recognition', that describe the characters in invariant terms with considerations of topological and morphological type. The method makes it possible of reading English numerical characters having strong variations in form, size and position.

A matrix of simple, identical intercommunicating cells is the core of a pattern classifier described by Glucksman [10]. A test sequencer tests the matrix, on which the pattern is mapped, by following a binary decision tree. The result of each test determines which of two tests will follow. Each test eliminates classes of patterns from the total set of classes. The process leads to one final class at the end of a sequence of such tests.

Yau and Yang [11] suggested simple template-matching pattern recognition technique by using any general purpose associative memory.

A.W. Holt in a paper [12] classifies the character recognition machinery with a

minimum reference to the specific components used. According to Holt, the job of all such machines is to convert a set of data having high information into a character name having a much lower information content. In his paper he gave a description of single stage, two stage and three stage character recognition machines. He is the first man to use stage concept in classifying character recognition machinery.

Fu, a pioneer in pattern recognition field, in his many papers discusses the different forms of recognition techniques. In a paper [4] he discussed in detail the non-parametric as well as parametric (Bayes) classification methods. The paper also includes a discussion on sequential decision model for pattern classification.

In another paper [13] Fu explores the topics on syntactic approach. Here a detailed discussion on selection on pattern primitives, pattern grammar including special grammar and syntax analysis as recognition procedure could be found.

M.A. Sattar and S.M. Rahman [14] in their paper discuss different problems of recognition of printed Bangla characters by applying the Template matching method. They discussed the optimum number of training set for designing the templates and the mismatch threshold to be used to avoid the problem of mis-classification.

W.W. Stallings describes [15] an approach to chinese character recognition, based on a formal model of the pictorial structure of chinese characters. He developed a program which produces a description of a character on two levels : i) the internal structure of each connected part of the character and ii) the arrangement in two dimensions of the connected parts. The program generates a numeric code from the structural representation of a character and is used for recognition.

A group of researchers at the Hitachi. Ltd. in Japan have developed a recognition technique based on the projection of Chinese characters on to the X and Y axes [16]. The authors observed that Chinese characters are typified mainly by the presence of vertical and horizontal strokes in specified positions and that this form is almost always well preserved in their projected profiles. Recognition is achieved by comparing the amplitude spectra of the profiles, the absolute value of the Fourier transform of the profiles, to those of known characters.

M.A. Sattar in his thesis [3] uses the decision-theoretic pattern recognition approach for recognizing Bangla alpha-numerics. The scheme uses the 'template-matching' technique for recognition. The input pattern (with unknown classification) is compared with a set of templates or prototypes, one for each character, designed previously and stored in machine and the classification is based on a good match with the template.

Md. Mashuqur Rahman in his thesis [17] suggested a syntactic pattern recognition scheme for recognizing English alphabetic characters of block capital type. He developed a FORTRAN IV program to work in steps : first for drawing the outline of the character, next for selecting nodes, third for recognizing morphs in between successive nodes, next for converting characters into numeric codes representing the relationship among the morphs which build the character,and finally for recognizing the character by matching the code with a dictionary of codes.

1.8 SCOPE OF THE PRESENT WORK.

It has already been mentioned that one of the main aspect of pattern recognition is to communicate with the computing machines in the natural mode of communication (the human voice and handwritten or typed script). Research works are going on in different ways of man-machine communication such as speaker recognition, speech analysis, finger print identification, character recognition etc. Character recognition has been receiving considerable attention as the result of phenomenal growth of the office automation and the need for translating human language into machine language.

With the rapid growth and advancement of the use of computers in Bangladesh, the use of our mother tongue Bangla in computers is being much talked about and much research is being done. As we find in many cases, the problem of input of Bangla materials to computers is time consuming and error prone. One answer to the problem would be the development of a practical Bangla character recognition method, towards which the effort of this project is directed.

The objective of this research is the development of a method for the recognition of printed Bangla characters using the syntactic method. Most workers, in the field of character recognition, followed decision-theoretic approach which is generally attractive only for classification of patterns ignoring structural information. Syntactic approach has been chosen for the present work since this method classifies the pattern as well as gives a structural description of it. Structural information of a character is important if we want to recognize a character irrespective of its size.

A computer program has been developed to work in steps:

a. Acquisition of the printed Bangla character to be recognized by a scanner.

b. Segmentation of the character.

c. Recognition of the pre-defined morphs (small basic elements) in the character.

d. Numeric code generation based on identified morphs.

e. Comparison of the numeric codes with the pre-stored codes (in a dictionary) for different Bangla characters and recognition of the input character.

CHAPTER TWO

SYNTACTIC METHOD

Observers are not led by the same physical evidence to the same picture of the universe unless their linguistic backgrounds are similar or can in some way be calibrated.

Benjamin Lee Whorf

CONTENT

CHAPTER 2. SYNTACTIC METHOD

2.1 Introduction 21

2.2 Syntactic Pattern Recognition System 25

2.3 Preprocessing Techniques 28

2.3.1 Encoding and Approximation 28

2.3.2 Filtering, Restoration and Enhancement 29

2.4 Picture Segmentation 29

2.4.1 Characteristics Feature Thresholding and Clustering 31

2.4.1.1 Thresholding 31

2.4.1.2 Clustering 33

2.4.2 Edge Detection 33

2.4.2.1 Parallel Techniques 34

2.4.2.2 Sequential Techniques 34

2.4.3 Region Extraction 35

2.4.3.1 Region Merging 35

2.4.3.2 Region Dividing 36

2.4.3.3 Region Merging and Dividing 36

2.5 Selection of Pattern Primitives 36

2.6 Primitive Recognition 39

2.7 Pattern Grammar 40

2.8 Recognition Procedure 42

CHAPTER 2

SYNTACTIC METHOD

2.1 INTRODUCTION.

Most of the development in pattern recognition research during the past two decade deal with the decision-theoretic approach and its applications. In some pattern recognition problems, the structural information that describes each pattern is important, and the recognition process includes not only the capability of assigning the pattern to a particular class (to classify it), but also the capacity to describe aspects of the pattern which make it ineligible for assignment to another class. A typical example of this class of recognition problem is picture recognition or, more generally speaking, scene analysis. In this class of recognition problems, the patterns under consideration are usually quite complex and the number of features required is often very large which make the idea of describing a complex pattern in terms of a (hierarchical) composition of simpler subpatterns very attractive. Also, when the patterns are complex and the number of possible descriptions is very large it is impractical to regard each description as defining a class (for example, in fingerprint and face identification problems, recognition of Bangla characters, etc.).

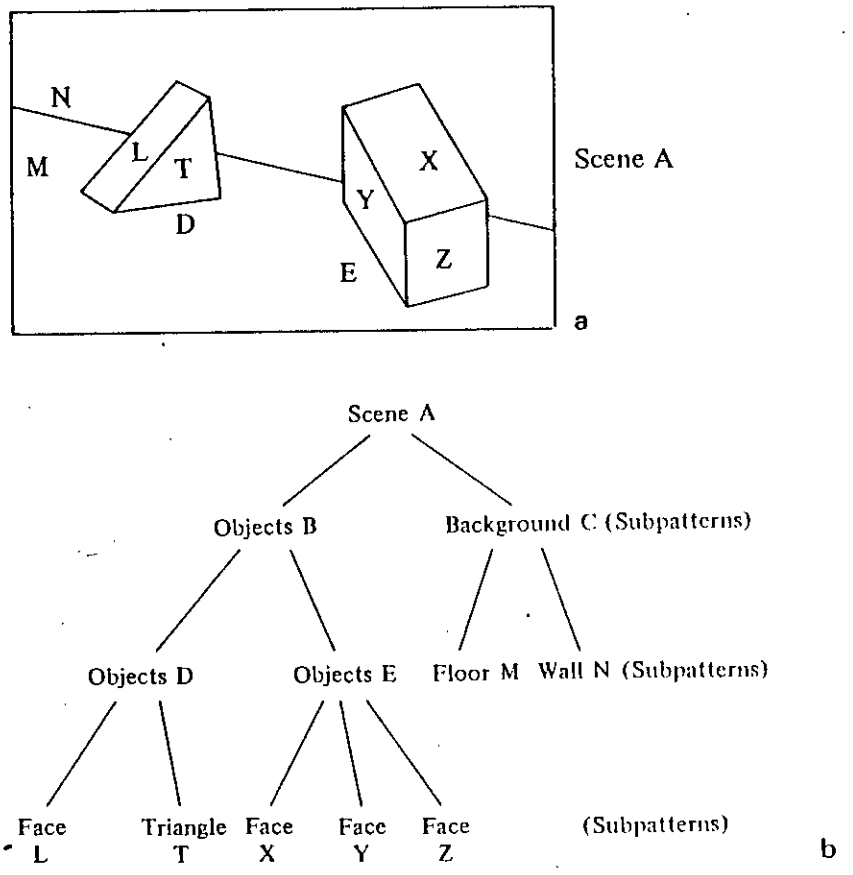


Fig. 2.1 Tree representation of a simple scene.

Consequently, the requirement of recognition can be satisfied only by a description for each pattern rather than the simple task of classification.

For example, the pictorial pattern shown in Fig. 2.1a can be described in terms of hierarchical structure shown in Fig. 2.1b.

In order to represent the hierarchical (tree-like) structural information of each pattern, that is, a pattern described in terms of simpler subpatterns and each simpler subpattern again described in terms of even simpler subpatterns, etc., the syntactic

or linguistic approach has been proposed [18]. This approach draws an analogy between the (hierarchical) structure of patterns and the syntax of languages. Patterns are specified as being built up out of subpatterns in various ways of composition, just as phrases and sentences are built up by concatenating words, and words are built up by concatenating characters. Evidently, for this approach to be advantageous, the simplest subpatterns selected, called *pattern primitives*, should be much easier to recognize than the patterns themselves. The *language* that provides the structural description of patterns in terms of a set of pattern primitives and their composition operations is sometimes called the *pattern description language*. The rules governing the composition of primitives into patterns are usually specified by the so-called *grammar* of the pattern description language (PDL). After each primitive within the pattern is identified, the recognition process is accomplished by performing a syntax analysis, or, parsing of the *sentence* describing the given pattern to determine whether or not it is syntactically (or grammatically) correct with respect to the specified grammar.

The syntactic approach to pattern recognition provides a capability for describing a large set of complex patterns by using small sets of simple pattern primitives and of grammatical rules. Of course, the practical utility of such an approach depends on our ability to recognize the simple pattern primitives and their relationships, represented by the composition operations.

The various relations or composition operation defined among subpatterns can usually be expressed in terms of logical and/or mathematical operations. For example, if we choose 'concatenation' as the only relation (composition operation)

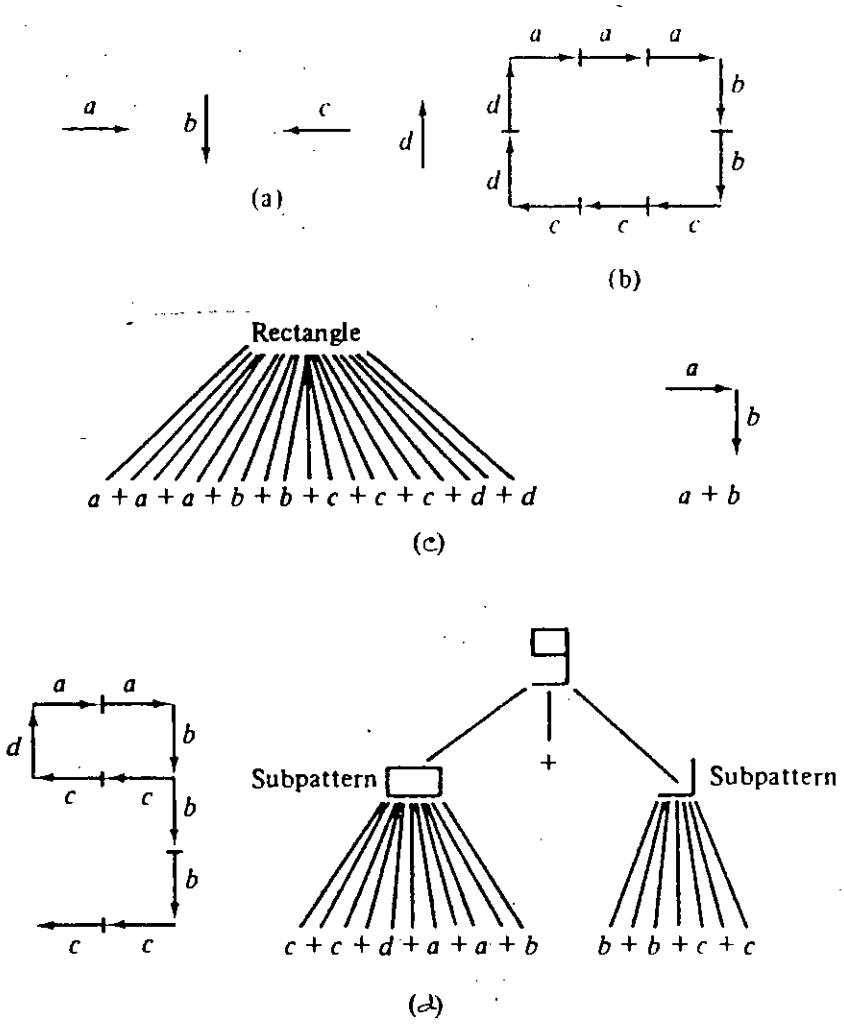


Fig. 2.2 (a) Pattern primitivrs; (b) a rectangle; (c) structural description of the rectangle in (b); (d) pattern 9 and its structural description.

used in describing patterns, then for the pattern primitives shown in Fig. 2.2(a) the rectangle in Fig. 2.2(b) would be represented by the string *aaabbccdd*. More explicitly, if we use '+' for the 'head-to-tail concatenation' operation, the rectangle in Fig. 2.2b would be represented by the string $a+a+a+b+b+c+c+c+d+d$, and its corresponding tree-like structure would be that shown in Fig. 2.2(c). Similarly, a slightly more complex example is given in Fig. 2.2(d) using pattern primitives

in Fig. 2.2(a). It is noted that, in this case, both the primitives and the relation between them are represented by the symbols in the string.

2.2 SYNTACTIC PATTERN RECOGNITION SYSTEM.

A syntactic pattern recognition system can be considered as consisting of two major parts: analysis and recognition, The *analysis* part consists of primitive selection and grammatical (or structural) inference, and the *recognition* part consists of preprocessing, segmentation or decomposition, primitive (and relation) recognition, and syntax analysis. A simple block diagram of the system is shown in Fig. 2.3.

The functions of preprocessing includes:

i) pattern encoding and approximation, and

ii) filtering, restoration and enhancement.

An input pattern is first coded or approximated by some convenient form for further processing. For example, a black-and-white picture can be coded in terms of a grid (or a matrix) of 0's and 1's. To make the processing in the later stages of the system more efficient, some sort of *data compression* is often applied at this stage. Then, techniques of filtering, restoration and/or enhancement will be used to clean the noise, to restore the degradation, and/or to improve the quality of the coded patterns. At the output of the preprocessor, we have patterns with reasonably 'good quality'.

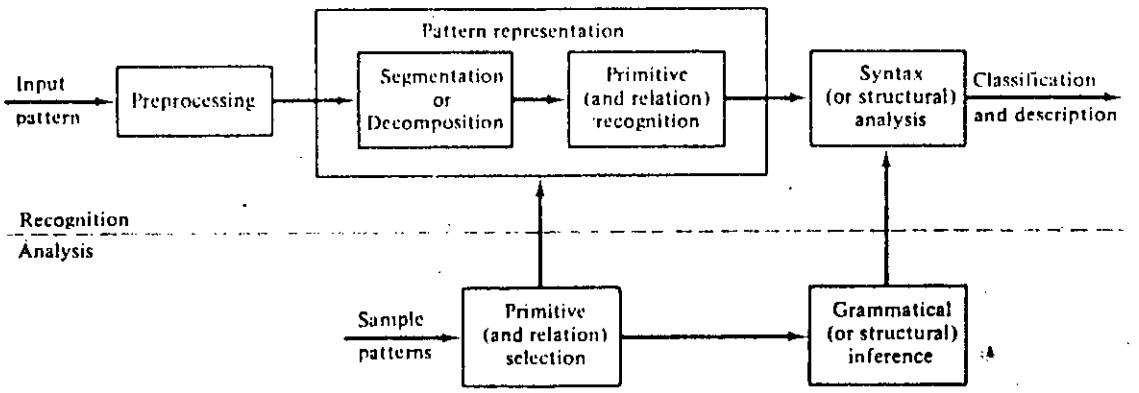


Fig. 2.3 Block diagram of a syntactic pattern recognition system.

Each preprocessed pattern is then represented by a language-like structure (for example, a string or a graph). The operation of this pattern-representation process consists of

i) pattern segmentation or decomposition

ii) primitive (and relation) recognition.

In order to represent a pattern in terms of its subpatterns, we must segmentize the pattern and in the meantime, identify the primitives and relations in it. Each pattern is now represented by a set of primitives with specified syntactic operations. For example, in terms of 'concatenation' operation, each pattern is represented by a string of (concatenated) primitives.

The decision as to whether or not the representation (pattern) is syntactically correct (i.e., belongs to the class of patterns described by the given syntax or grammar) will be performed by the *syntax analyzer* or *parser*. Conceptually, the simplest form of recognition is probably *template-matching*. The string of primitives representing an input pattern is matched against strings of primitives representing each prototype or reference pattern. Based on a selected *matching* or *similarity* criterion, the input pattern is classified in the same class as the prototype pattern that is the 'best' match to the input. In between there are a number of intermediate approaches. For example, a series of tests can be designed to test the occurrence or nonoccurrence of certain subpatterns (or primitives) or certain combinations of subpatterns or primitives. The result of the tests (e.g., through a table-look-up, decision tree, or logical operation) is used for a classification decision. The selection of an appropriate approach for recognition usually depends on the problem requirement. If a complete pattern description is required for recognition, parsing is necessary. Otherwise, a complete parsing could be avoided by using other, simpler approaches to improve the efficiency of the recognition process.

In order to have a grammar describing the structural information about the class of patterns under study, a grammatical inference machine is required which can infer a grammar from a given set of training patterns in language-like representations. This machine's function is analogous to the 'learning' process in a decision-theoretic pattern recognition system. The structural description of the class of patterns under study is learned from the actual sample patterns from that class. The learned description, in the form of a grammar, is then used for pattern de-

scription and syntax analysis (see Fig. 2.3). A more general form of learning might include the capability of learning the best set of primitives and the corresponding structural description for the class of patterns concerned.

2.3 PREPROCESSING TECHNIQUES.

The functions of preprocessing include (1) pattern encoding and approximation; and (2) filtering, restoration, and enhancement.

2.3.1 Encoding and Approximation.

For the purpose of processing by digital computers, patterns are usually digitized (in time or in space) first, and then represented by more compact forms (data compression) using coding or approximation schemes. Consider that a pattern is a one-dimensional (e.g., wave form) or two-dimensional (e.g., picture) function. Based on the sampling theorems, we know that a given function can be reconstructed exactly by the samples taken at a finite set of points.

Quantization is another form of approximation. In this case, the actual value of the function is approximated by only a finite set of discrete values, called *quantization levels*. After a pattern is sampled and quantized, the quantized value at each sample point is usually coded and is used to represent (approximate) the pattern function at the point. Let there be m possible quantization levels. It will take at least $log_2 m$ bits to represents any given level. If the probability of occurrence of

each possible level is known, a more efficient code can be constructed to reduce the average of bits representing each level. The basic idea in constructing such codes is to use short codes for the commonly occurring quantization levels.

2.3.2 Filtering, Restoration and Enhancement.

Time-invariant and position-invariant (or shift-invariant) operations can be used to *filter* a pattern in order to detect a given pattern in it, to *restore* a pattern that has been degraded by approximation or other such operations, and to *smooth* or *enhance* a pattern to improve its quality. Such operations can be implemented not only by conventional digital computers, but also by special-purpose electro-optical devices. Let F be a set of pattern functions and T be a translation operation in time or in space defined on F. An operation ϕ is called *time-invariant* (or *position-invariant*) if it commutes with every T in time (or in space). That is

$$\phi(T(f)) = T(\phi(f))$$

for all $f \in F$. Operations of this type have the property that their effect on a point of the pattern function does not depend on its time or position in the pattern.

2.4 PICTURE SEGMENTATION.

There is no universal method of segmenting a picture into subpictures. Different types of subpictures can be useful, depending on the type of description that is

required. During the past decade, many image segmentation techniques have been proposed [18]. These segmentation techniques can be categorized into three classes:

1) Characteristics feature thresholding or clustering,

2) Edge detection,

3) Region extraction.

One way to define image segmentation is as follows [18]. Let X denote the grid of sample points of a picture, that is, the set of pairs

$$\{i, j\} \quad i = 1, 2, \ldots, N, \quad j = 1, 2, \ldots, M$$

where N and M are the number of pixels in the x and y directions, respectively. Let Y be a non-empty subset of X consisting of contiguous picture points. Then a uniform predicate $P(Y)$ is one that assigns the value true or false to Y, depending only on properties related to the gray level or brightness matrix $f(i, j)$ for the points of Y. Furthermore, P has the property that if Z is a non-empty subset of Y, then $P(Y) =$ true implies always $P(Z) =$ true.

A segmentation of the grid X for a uniformity predicate P is a partition of X into disjoint non-empty subsets $X_1, X_2, \ldots, X_N$ such that:

1. $\bigcup_{i=1}^{N} X_i = X$.
2. $X_i, \quad i = 1, 2, \ldots, N$ is connected.
3. $P(X_i) =$ true $\quad$ for $i = 1, 2, \ldots, N$.
4. $P(X_i \cup X_j) =$ false $\quad$ for $i \neq$ j, where X_i and X_j are adjacent.

The foregoing conditions can be summarized as follows. The first condition implies that every picture point must be in a region. This means that the segmen-

tation algorithm should not terminate until every point is processed. The second condition implies that regions must be connected (i.e., composed of contiguous lattice points). The third condition determines what kind of properties the segment regions should have, for example, uniform gray levels. The fourth condition express the maximality of each region in the segmentation.

Almost all image segmentation techniques proposed so far are ad hoc in nature. There are no general algorithms that will work for all image. An image segmentation problem is basically one of psychophysical perception, and therefore not susceptible to a purely analytical solution. Any mathematical algorithm must be supplemented by heuristics, usually involving semantics about the class of pictures under consideration. Quite often, one must go beyond simple heuristics, and the introduction of a priori knowledge about the picture is essential. Most segmentation algorithms are either based on the concept of similarity (e.g., characteristic feature clustering algorithms) or discontinuity (e.g., edge detection algorithms).

2.4.1 Characteristics Feature Thresholding and Clustering.

2.4.1.1 Thresholding

Characteristic feature thresholding is a technique widely used in picture segmentation. In its most general form, *thresholding* is described mathematically as

$$S(x,y) = k \quad if \quad T_{k-1} \leq f(x,y) < T_k \qquad k = 0,1,2,\ldots,m$$

where (x,y) represents the x and y coordinates of a picture element or *pixel*; $S(x,y)$, $f(x,y)$ are the segmented and the characteristic feature (e.g., gray level) functions of (x,y), respectively; $T_0,\ldots,T_m$ are threshold values with T_0 equal to

the minimum and T_m the maximum; and m is the number of distinct labels assigned to the segmented picture. A threshold operator T can be viewed as a test involving a function T of the form

$$T(x, y, N(x, y), f(x, y))$$

where $N(x, y)$ denotes some local property of the point (x, y) (e.g., the average level over some neighborhood). Depending on the functional dependencies of the threshold operator T, thresholding can be divided into three types. When T depends only on $f(x, y)$, the threshold is called *global*. If T depends on both $f(x, y)$ and $N(x, y)$, it is called a *local* threshold. If T depends on the coordinate values x, y as well as on $f(x, y)$ and $N(x, y)$, it is called *dynamic* threshold.

There are a number of global threshold selection schemes. Some are based on the characteristics feature (e.g., gray level) histogram, others are based on local properties such as the gradient or Laplacian of a picture. For a picture consisting of object and background where the percent of the object area is known, "p-tile" method has been suggested [18], which chooses as a threshold the gray level that most closely corresponds to mapping at least $(1 - p)\%$ of the gray levels into the object. If, for example, dark objects occupy 20% of the picture area, then the image should be thresholded at the 80th percentile, or more precisely, at the largest gray level allowing at least 20% of the picture points to be mapped into the object. This method is not applicable if the object area is unknown or varies from picture to picture.

For segmenting pictures of white blood cells, thresholds at the valleys on the gray-level histogram has been chosen. This technique, called the *mode method*,

involves smoothing of the histogram into a predetermined number of peals (modes) and placing thresholds at the valleys between peaks. The mode method has the advantage that it minimizes the probability of misclassifying an object pixel as background, or vice versa.

2.4.1.2 Clustering

Clustering of characteristics feature applied to picture segmentation is multi-dimensional extension of the concept of thresholding. Typically, two or more characteristic features are used and each class of regions is assumed to form a distinct cluster in the space of these characteristic feature. A clustering method is used to group the point in the characteristic feature space into clusters. These clusters are then mapped back to the original spatial domain to produce a segmentation of a picture. The characteristic features that are commonly used in picture segmentation by clustering not only include gray through different filters, but may include any feature that one thinks ought to be helpful to the segmentation problem. The reason one wants to use two or more characteristic features to perform image segmentation is that sometimes there are problems not resolvable with one feature that can be resolved with two or more features.

2.4.2 Edge Detection.

Edge detection is a picture segmentation technique based on the detection of discontinuity. A *edge* or *boundary* is the place where there is a more or less abrupt change in gray level. Some of the motivating factors of this approach are : (1) most of the information of a picture lies on the boundaries between different regions

and (2) biological visual systems appear to make use of edge detection, but not of thresholding. Edge detection techniques can be categorized into two categories, parallel and sequential.

2.4.2.1 Parallel Techniques

By a *parallel solution* to the edge detection problem, it is meant that the decision of whether or not a set of points is on an edge is made on the basis of the gray level of the set and some set of its neighbors; but the decision is not dependent on first deciding if other sets of points lie on an edge. So the edge detection operator in principle can be applied simultaneously everywhere in the picture.

2.4.2.2 Sequential Techniques

A sequential edge detection, the result at a point is contingent upon the results of the operator at previously examined points. the major components of a sequential edge detection procedure are:

1. *Choosing a good initial point*: The performance of the entire procedure will depend upon the choice of a good starting point.
2. *The dependence structure*: How do the results obtained at previously examined points affect both the choice of the next point to be examined and the result at the next point ?
3. *A termination criterion*: There must be a way for the procedure to determine that it is finished.

It should be noted that the definitions of parallel techniques and sequential techniques are stated with respect to edge element extraction. To produce a closed edge or boundary, the edge elements extracted have to be connected together to

form closed curves. Although the method of generating edge elements is parallel in nature, the method of connecting these extracted edge elements could be sequential in nature, depending on the method used; this method will be referred to as a *parallel scheme*. Parallel edge detection schemes can be broken down generally into two steps: (1) edge element extraction and (2) edge element combination (also called *streak* boundary formation). Edge element combination consists of eliminating false edge elements and merging the edge elements into longer edge segments called streaks, eliminating false streaks, combining the streaks into boundaries, and eliminating false boundaries. Edge element combination is generally carried out by three classes of techniques: (a) heuristic search and dynamic programming, (b) relaxation, and (c) line and curve fitting.

2.4.3 Region Extraction.

Another way of doing image segmentation instead of finding boundaries of regions is to divide the image into regions. Region extraction techniques can be broken down into three categories: (1) region merging, (2) region dividing and (3) a combination of region merging and dividing.

2.4.3.1 Region Merging

Muerle and Allen [25] used a regional neighbor search method to merge regions of similar properties. Pavlidis [26] partitioned a picture into a collection of one-dimensional strips, divided the strips into segments and merged the segments with similar approximation coefficients.

2.4.3.2 Region Dividing

One way of doing picture segmentation by region extraction is the region dividing approach. Robertson et al. [27] used a mean vector of gray levels of a multispectral image to perform region dividing. Klinger [28] proposed to use regular decomposition for image segmentation.

2.4.3.3 Region Merging and Dividing

Horowitz and Pavlidis [29] approached the problem using a 'split and merge' principle. Regions were described in terms of an approximating function. They merged adjacent regions having similar approximations and split those regions that had large approximating errors.

2.5 SELECTION OF PATTERN PRIMITIVES.

The first step in formulating a syntactic model for pattern description is determination of a set of primitives in terms of which the patterns of interest may be described. This determination will be largely influenced by the nature of the data, the specific application in question and the technology available for implementing the system. There is no general solution for the primitive selection problem at this time. The following requirements usually serve as a guide for selecting pattern primitives.

1. The primitives should serve as basic pattern elements to provide a compact but adequate description of the patterns in terms of the specified structural

relations (e.g., the concatenation relation).

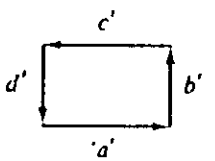
2. The primitives should be easily extractable or recognisable by existing nonsyntactic (e.g., decision-theoretic) methods, since they are considered to be simple and compact patterns and their structural information is not important.

For speech patterns, phonemes are naturally considered as 'good' set of primitives with the concatenation relation. Similarly, strokes have been suggested as primitives in describing Bangla characters. However, for general pictorial patterns, there is no such "universal picture elements" analogous to phonemes in speech or strokes in Bangla characters. Sometimes, in order to provide an adequate description of the patterns, the primitives should contain the information that is important to the specific application in question. For example, if size (or shape or location) is important in the recognition problem, then the primitives should contain information relating to size (or shape or location), so that patterns from different classes are distinguishable by whatever method is to be applied to analyze the descriptions. This requirement often results in a need for semantic information in describing primitives. The following simple example is used to illustrate that, for the same data, different problem specifications would results in different selection of primitives.

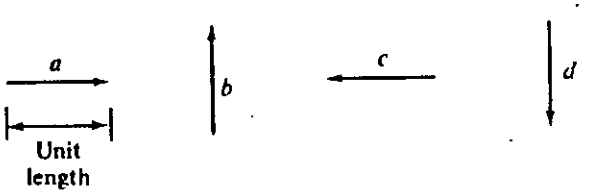
Suppose that the problem is to discriminate rectangles (of different sizes) from nonrectangles. The following set of primitives is selected :

a'	0°	horizontal line segment
b'	90°	vertical line segment
c'	180°	horizontal line segment
d'	270°	vertical line segment

The set of all rectangles (of different sizes) is represented by a single sentence or string $a'b'c'd'$.



If the problem is also to discriminate rectangles of different sizes, the foregoing description would be inadequate. An alternative is to use unit-length line segments as primitives:



The set of rectangles of different sizes can then be described by the language

$$L = \{a^n b^m c^n d^m \mid n, m = 1, 2, \ldots\}$$

Requirement 2 may sometimes conflict with requirement 1 because the primitives selected according to requirement 1 may not be easy to recognize using existing simple techniques. On the other hand, requirement 2 could allow the selection of quite complex primitives as long as they can be recognized. With more complex primitives, similar structural descriptions (e.g., simple grammars) of the patterns

could be used. This trade-off may become quite important in the implementation of the recognition system.

2.6 PRIMITIVE RECOGNITION.

Since primitives are basically simple components of a pattern and contain very little structural information, their recognition is often just a simple classification problem based on a small number of features or attributes. Template matching or multiple thresholding of various feature values can be applied to primitive recognition. Templates are designed for each type of (class of) primitives. A segmented part or component of the pattern is matched against the templates, and is recognized as the primitive of which the template 'best' matches the input component. Such a matching could carried out it terms of raw data of pattern or a selected set of features. Correlation and various distance and error criteria have been suggested as matching criteria.

Recognition of a primitive can also be made on the basis of a logical decision from the comparisons of feature measurements taken from the input component with a set of prespecified thresholds. For example, straight-line segments can be classified into different classes by computing their lengths with different thresholds and/or their slopes with prespecified thresholds.

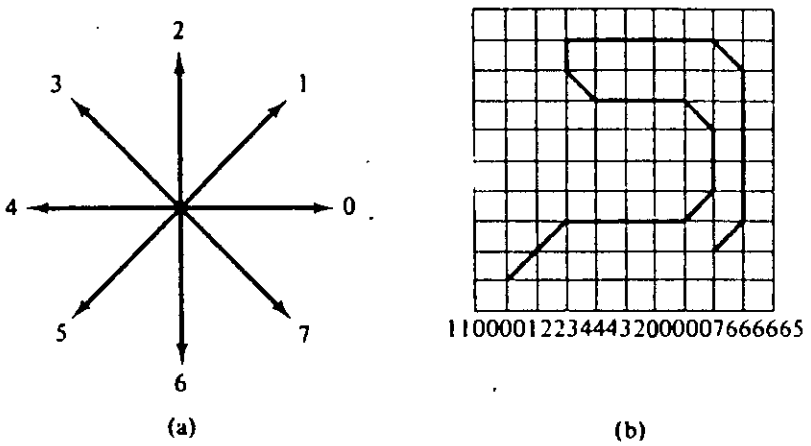


Fig. 2.4 Freeman's chain code

Freeman's Chain Code [18]: Straight-line with fixed length and slopes within different ranges have been proposed as primitives in describing line drawings. Typically, eight primitives (the octal code) are used as shown in Fig.2.5(a). A line pattern represented by a string of these eight primitives is shown in Fig. 2.4(b).

2.7 PATTERN GRAMMAR.

Let us assume that a satisfactory solution of the primitive selection problem is available for a given application. The next step is the construction of a grammar (or grammars) that will generate a language (or languages) to describe the patterns under study. Ideally, it would be nice to have a grammatical inference machine that

would infer a grammar from a set of given strings describing the patterns under study. Unfortunately, such a machine has not been available except for some very special cases. In most cases so far, the designer constructs the grammar based on the a priori knowledge available and his or her experience.

It should be remarked that a grammar is most appropriate for description when the pattern of interest is built up from a small set of primitives by recursive application of a small set of production rules. Also, primitive selection and grammar construction should probably be treated simultaneously rather than in two different stages. There is no doubt that a different selection of pattern primitives will result in a different grammar for the description of a given set of patterns. Sometimes, a compromise is necessary in order to develop a suitable grammar. Referring to the previous discussion, it is evident that the grammar which generates $L = \{a^n b^m c^n d^m \mid n, m = 1, 2, \ldots\}$ will be much more complex than the grammar generating $a^l b^l c^l d^l$.

In addition to (1) the trade-off between the language descriptive power and the analysis efficiency and (2) the compromise sometimes necessary between the primitives selected and the grammar constructed, the designer should also be aware of need to control the excessive strings generated by the constructed grammar. The number of pattern strings available in practice is always limited. However, in most cases the grammar constructed would generate a large or infinite number of strings. It is hoped that the excessive strings generated are similar to the available pattern strings. Unfortunately, this may not be true, since the grammar, in many cases, is constructed heuristically. The problem may become very serious when the excessive

strings include some pattern strings that should belong to other classes. In this case, adjustments should be made to exclude these strings from the language generated by the constructed grammar.

2.8 RECOGNITION PROCEDURE.

After a grammar is constructed to generate a language that would describe the patterns under study, the next step is to design a recognizer that will recognize the patterns (represented by strings) generated by the grammar. A straight-forward approach might be to construct a grammar for each class of patterns. The recognizer designed for a particular grammar will recognize only the patterns in the class corresponding to this grammar. For example, corresponding to m classes of patterns $w_1, w_2, \ldots, w_m$, we can construct m grammars $G_1, G_2, \ldots, G_m$, respectively, such that the strings generated by the grammar G_i would represent the patterns in class w_i. For an unknown pattern described by the string x, the problem of recognition is essentially reduced to the answer to the question:

$$\text{Is} \quad x \in L(G_i) \qquad \text{for} \quad i = 1, 2, \ldots, m?$$

The process that would result in an answer to such a question with respect to a given grammar is, in general, called *syntax analysis* or *parsing*. A block diagram for the recognizer is shown in Fig. 2.5.

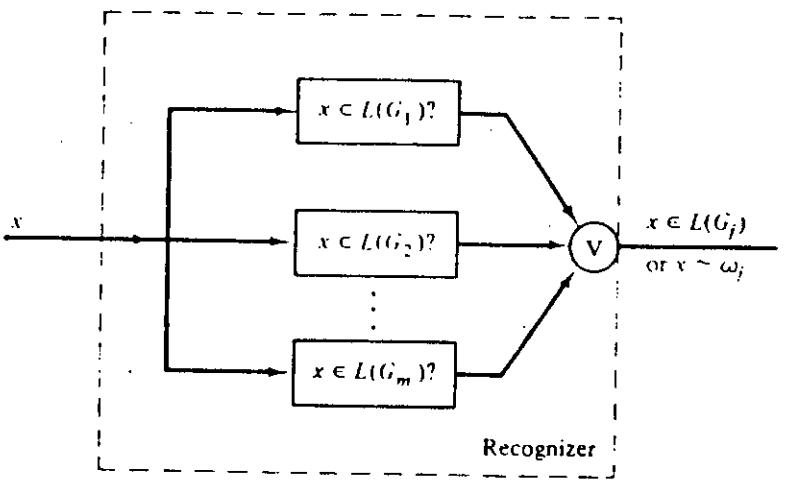


Fig. 2.5 Block diagram of the syntactic recognizer.

Presumably, if different patterns are uniquely represented by different strings, only one answer to the m questions "Is $x \in L(G_i)$ for $i = 1, 2, \ldots, m$?" will be yes; the rest will be no. If the answer is $x \in L(G_j)$, then x is recognized from the class w_j. If all the answers are no, the pattern x will be rejected; that is, x is not accepted as from any one of the m classes.

A very simple form of implementing the recognizer is to match the input string x with some prototype or reference strings from each class in terms of pattern primitives. The matching of primitives can be performed in parallel or sequentially. x will be recognized as from the same class of the reference string that 'best' matches x. In this case, either a perfect match or an appropriately selected matching cri-

terion is required. The choice of reference strings is of course also very important. This approach, although it seems less flexible, is simple and speedy in processing. However, the syntactic (hierarchical structural) information of the string has really not been sufficiently utilized in the recognition and the approach is useful only when appropriate reference strings and a meaningful matching criterion can be determined.

An obvious extension of this approach to include syntactic information is to match the graph (e.g., relational graph) describing the unknown pattern (or subgraphs) against various prototype or reference graphs, or to match the derivation tree (parse) of the unknown string x (or subtrees) against reference parses. It is also possible, for some problems, to set up a group of tests that determine whether or not a pattern has certain attributes and/or what the actual values of the attributes are. Based on the outcome of these tests, the pattern can be assigned to one of the predetermined classes.

The process or the algorithm of performing the recognition is called *syntax analysis*, which uses, in general, a nondeterministic procedure. The output from the analyzer usually includes not only the decision of accepting the string generated by the given grammar, but also the derivation tree of the string, which, in turn gives the complete description of the pattern and its subpatterns.

CHAPTER THREE

DEVELOPMENT OF BANGLA CHARACTER RECOGNITION SYSTEM

One brought me happiness:

the other, recognition.

Anonymous.

CONTENT

CHAPTER 3. DEVELOPMENT OF BANGLA CHARACTER RECOGNITION SYSTEM

3.1 Introduction 45

3.2 Methodology of the Recognition Process 45

3.3 Input: Digitization 47

3.3.1 Scanner 48

3.4 Preprocessor 49

3.4.1 Get Letter 51

3.4.2 Get Letter Contour: Contour Tracing 53

3.4.3 Filtering 56

3.5 Extraction of Strokes (Morphs) 58

3.6 Encoding of Strokes (Morphs) 61

3.6.1 Selection of Strokes 61

3.6.2 Finding Stroke Code 61

3.7 Encoding of Characters 66

3.8 Recognition of Characters 67

3.8.1 Characters Having More Than One Code 67

3.8.2 Characters With Same Code 77

3.8.3 Development of a Dictionary 84

CHAPTER 3

DEVELOPMENT OF BANGLA CHARACTER RECOGNITION SYSTEM

3.1 INTRODUCTION.

This chapter reports on a study of the syntactic or structural approach to the recognition of Bangla characters. Using this approach, a scheme for automatic pattern recognition has been developed. A program has been written that analyzes Bangla characters; it produces a data structure which describes a character in terms of basic elements and the relationship among them. A routine has been developed for generating a numeric code from the structural representation. Recognition is achieved by building up a dictionary of character codes; the code for any new instance of a character can then be looked up in the dictionary.

3.2 METHODOLOGY OF THE RECOGNITION PROCESS.

To design the character recognition system the following methodological steps are followed:

a. **Digitization:** The input character is first to be transformed into binary form for the ease of handling by the computer.

b. **Contour Tracing:** The border line of the input character is traced to detect a transition from white to black.

c. **Filtering:** Techniques of filtering are used to clean the noise present in the character contour.

d. **Morph Extraction and Encoding:** Next task is to extract the morphs or strokes from the letter contour and to assign a code to each stroke.

e. **Character Encoding:** From the structural representation of a character, a numeric string type code is generated which uniquely identifies the character.

f. **Taking Decision:** Depending on the previous 'learning', the machine takes the decision to the class belongings of the character. A dictionary of codes is searched to find a matching code corresponding to the code generated from the input character. If the input character code matches uniquely with a code of the dictionary, then the classifier gives the decision that the character belongs to the matched class, otherwise, the decision is taken that the character cannot be recognized and it belongs to rejection class.

A schematic block diagram of the proposed methodology is shown in Fig. 3.1. In the subsequent sections the design procedure of each step mentioned above is discussed elaborately.

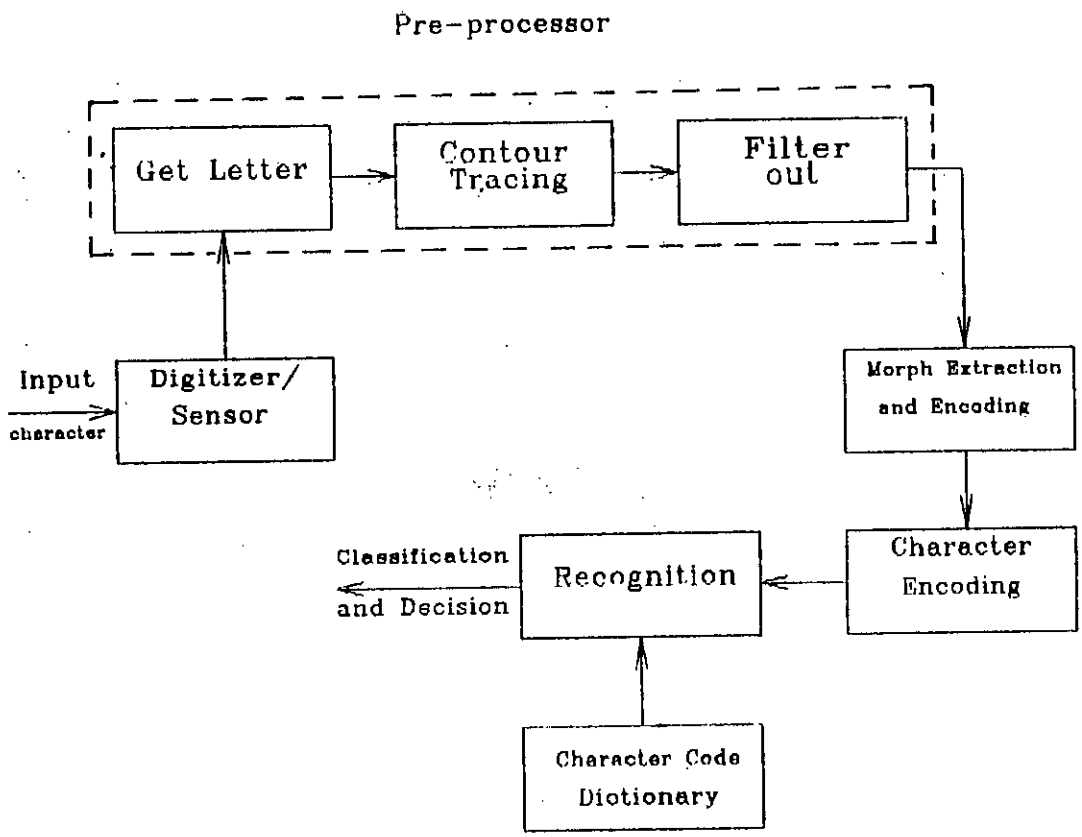


Fig. 3.1 Block diagram showing the steps of the developed Bangla character recognition system.

3.3 INPUT: DIGITIZATION.

To recognize a given character, at first it is to be digitized onto a matrix i.e., to be transformed into binary form for ease of handling by the computer. The *digitizer* is a device which coverts a physical sample to be recognized into *pattern vector*. It is often convenient, for recognition purpose, to arrange the pattern (character) in

the form of a pattern vector:

$$X = (x_1, x_2, x_3, \ldots, x_n)$$

where n is the number of measurements and component x_i of the vector X assumes the value 1 or 0, depending on the state of the ith position for a particular input.

The program operated on a representation of one character at a time. The representation is in the form of a matrix, whose entries have value 0 or 1 corresponding to black or white in the original picture. The matrix is obtained by means of a scanner. The printed characters used were taken from a number of different sources; the characters were all of roughly the same style but varied considerably in size. The resulting matrix is used as the database for the program.

3.3.1 Scanner.

The digitizer used is a scanner capable of scanning images of 10.5 cm(4.13 inches) wide. The maximum resolution of the scanned image is 8 dots per mm (200 dots per inch) in both directions which is equivalent to 840 dots per line. Though the scanner offers four encoding modes: three half-tone encoding and one black and white encoding, the black and white option is used to scan the characters. This mode converts the image into binary image: '0' for the presence of a black spot, '1' for the absence of any black spot. Fig. 3.2 shows a binary image of a character. The brightness of the scanned image can be controlled by the user. The resolution of the scanned image can also be controlled by another switch. At position 1, the resolution is lowest (50 dpi), whereas at position 4, the resolution is highest (200 dpi). The scanner is software controlled. The output of the scanner i.e., the binary

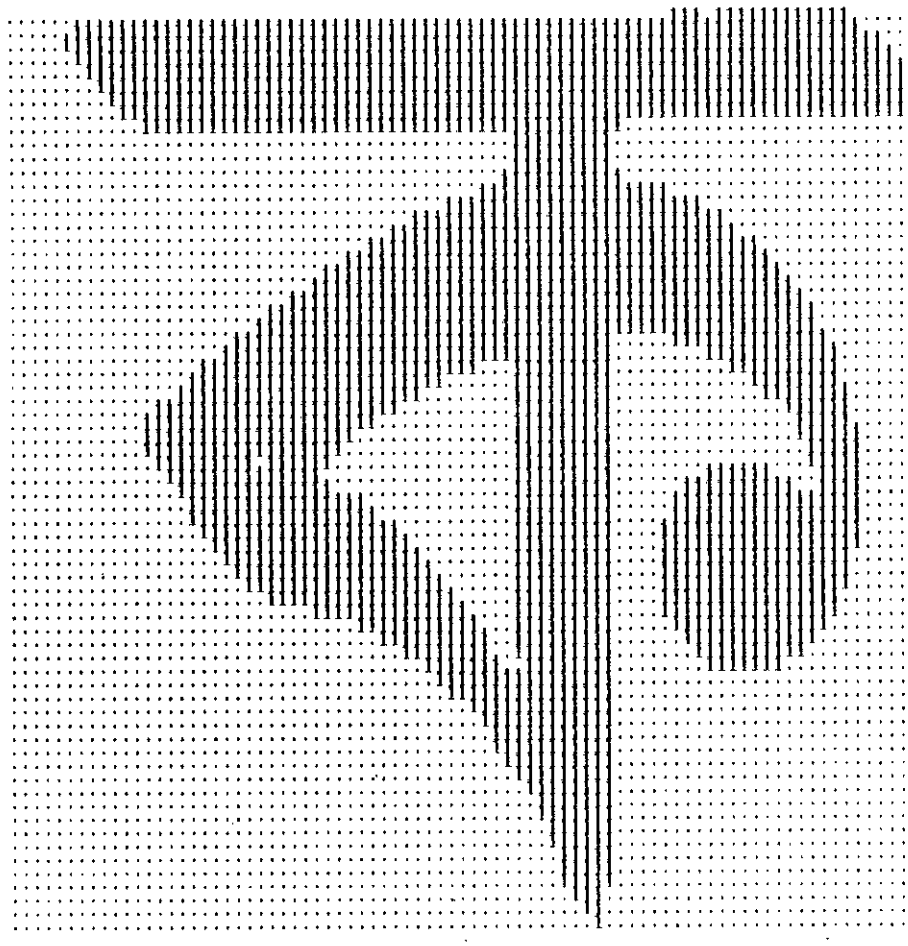


Fig. 3.2 Binary image of a sample character

image obtained by scanning a character is stored in the file LETTER.TMP for further use. The output of the scanner is shown in Fig. 3.3.

3.4 PREPROCESSOR.

The functions of preprocessing include

a) character encoding and approximation

b) contour tracing, and

c) filtering and enhancement

An input character is first coded or approximated in terms of a matrix of 0's and

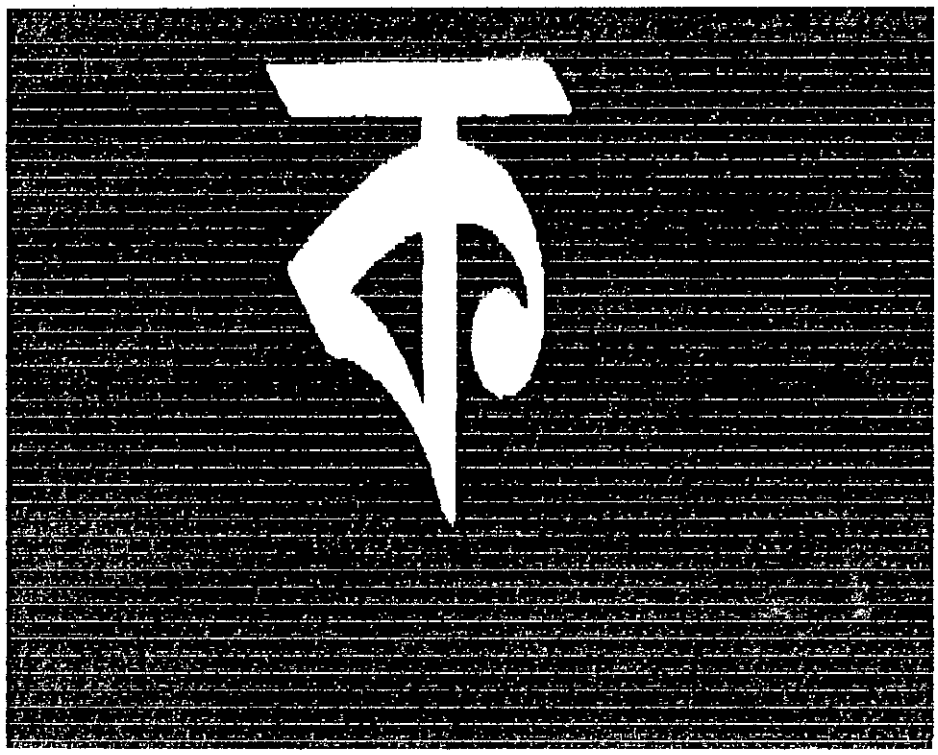


Fig. 3.3 The image of a sample character as obtained from the output of the scanner.

1's. Then the border line of the input pattern is traced to detect a transition from white to black. Then techniques of filtering and enhancement are used to clean the noise and to improve the quality of the contour of the pattern. At the output of the preprocessor, we have a letter contour of reasonably good quality.

A routine is written to get the actual area of the letter from the scanner output.

To construct the letter contour, another routine is written. In addition, one more function is written to clean the noise from the letter contour. It will be helpful to describe these first.

3.4.1 Get Letter.

At first the structure of the input character is read from the file LETTER.TMP to an array. This file is created by scanning the letter with the scanner. It contains the full screen bit map in $720/8 * 348$ bytes. From the full screen bit map, the task is to extract the area containing actual letter and to store it in a memory location. To do this, a routine *GetLetter* is employed.

GetLetter proceeds by scanning alternately from left to right and form top to bottom along various rows and columns of the input array. This continues until a black byte is encountered. A byte is taken as black if it has more than one black points. If a line containing black points has been found first, it is the top margin line and the scanning continues until the bottom margin line is found. Then a memory is allocated and the actual letter is transferred to that memory. Before transfer, the bit map of the letter is negated bit-wise.

Fig. 3.4 shows the output of the routine *GetLetter()* corresponding to the scanner output as shown in Fig. 3.3. The algorithm is formally described as follows [For source-coding refer to *GetLetter()* function in Appendix C]:

```
function GetLetter;
  begin
    RawLetter[SCREENHEIGHT][SCREENWIDTH];
    open input file;
    BlankLineNo:=0;
    for y:=1 to SCREENHEIGHT do
      begin
        BlackFound:=FALSE;
        for x:=1 to SCREENWIDTH do
          begin
            transfer each byte from the file to array RawLetter,
            negate bitwise before transfer;
            if a byte is significantly black then
                 {a byte is taken significantly black if it
                  has more than one black(1) points}
              begin
                BlackFound:=TRUE;
                update Left and Right margins;
              end;
          end;
        if BlackFound and Top has not been assigned then
          Top:=y;
        if BlackFound and BlankLineNo > 0 then
          BlankLineNo:=0;
        if not BlackFound and Top!=0 then
          if BlankLineNo < MAXBLANKLINE then
            increment BlankLineNo;
          else
            begin
              Bottom:= y - BlankLineNo;
              exit from the for-statement;
            end;
    end;
    LetterHeight:= Bottom-Top +1;
    LetterWidth := Right -Left+1;
    allocate memory and transfer actual area of
    the letter to that memory;
  end;
```

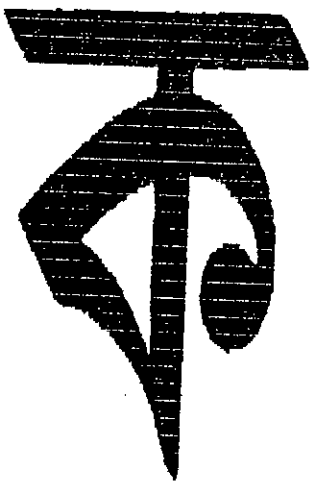


Fig. 3.4 The image after execution of GetLetter.

3.4.2 Get Letter Contour: Contour Tracing.

Contour tracing is the process of finding a series of black points on the boundary of a black region in a white field. The white to black transition points of the letter are obtained by scanning each line of the digitized character bytewise from left to right and from top to bottom and storing them in a memory location.

An array 'Transition' is used to increase the speed of the algorithm. With two or more consecutive 1's in the binary form of the offset, the array element is simply the binary representation of the offset having the leftmost '1' of consecutive 1's. For example, for the offset 14(= 00001110), the array element is 0x08(= 00001000).

Again, for 27(= 00011011), the array element is 0x12(= 00010010). Also a flag 'TransitionState' is used which indicates whether we are already in the middle of a black region, meaning transition started. If bit zero of the binary representation of the input character is '0', it indicates that the transition has not been started; if bit zero is '1', it is assumed that transition has been started. This flag with the help of the array gets the '0' to '1' transition points of consecutive bytes of the input character. To illustrate this, an example is shown in Fig. 3.5.

00011111 11101111	00010000 00001000
(a)	
00011100 11101111	00010000 10001000
(b)	

Fig. 3.5 Getting '0' to '1' transition points. (a) transition has been started. (b) transition has not been started.

In (a), since the contents of the first and second bytes are 00011111 and 11101111 respectively, from the array 00010000 and 10001000 can be obtained. Having the value '1', bit zero of the first byte indicates that transition from white to black has already been started. So bit seven of the second byte can never be '1'. It is changed to '0' by the 'TransitionState' flag. In (b), bit zero of the first byte is '0' and bit seven of the second byte is '1'. This indicates a '0' to '1' transition at the bit seven

position of the second byte, i.e., transition has not been started from the first byte or from the previous bytes. Hence bit seven of the second byte should keep at '1'. After tracing the border line, the character of Fig. 3.2 becomes as shown in Fig. 3.6.

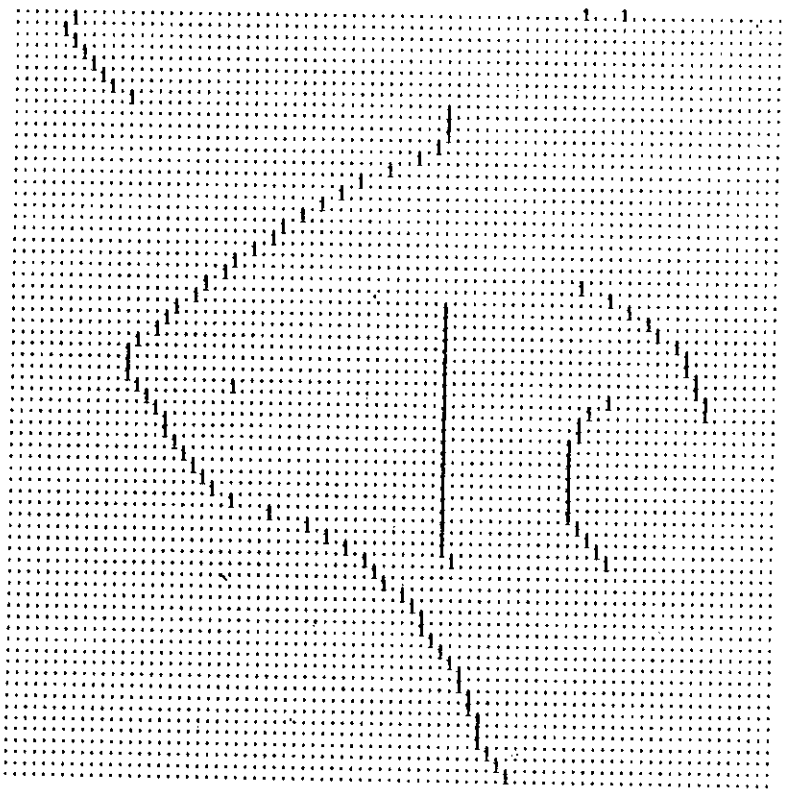


Fig. 3.6 Character ま, after its border line is traced

A general algorithm for contour tracing process is shown below. [For source coding refer to *Getlettercontour()* function in Appendix C].

```
function GetLetterContour;
  begin
    for y:=1 to LetterHeight do
      begin
        scan each line of the Letter from left to right
           and get the white(0) to black(1) transition;
        put '1' at the transition point;
        store them in the same memory location;
      end;
  end;
```

3.4.3 Filtering.

Next task towards pattern recognition is to filter out all isolated 1's (stray points) present in the contour map of the character to be recognized. An '1' is considered as to be isolated if there is no '1' at any one of its fourteen neighbor points as shown in Fig. 3.7.

```
000000000010          000000010000
000000100000          000001000000
001000000000          000100000000

      (a)                   (b)
```

Fig. 3.7 (a) An isolated '1', (b) an '1' having neighbor points.

An array 'FilterMask' is used to increase the speed of the filter out algorithm. Elements in the array are the filter mask for cleaning the noise. Each element consists of the binary form of the element offset with three 1's at the left and right side of the 1's. For example, element 9(=00001001) is 0x7F(=01111111). Again, element 20(=00010100) is 0xFF(=11111111). For elements with two or more consecutive 1's in the binary form of the offset, the array element is simply 0. For example, element 22(=00010110) is 0x00(=00000000).

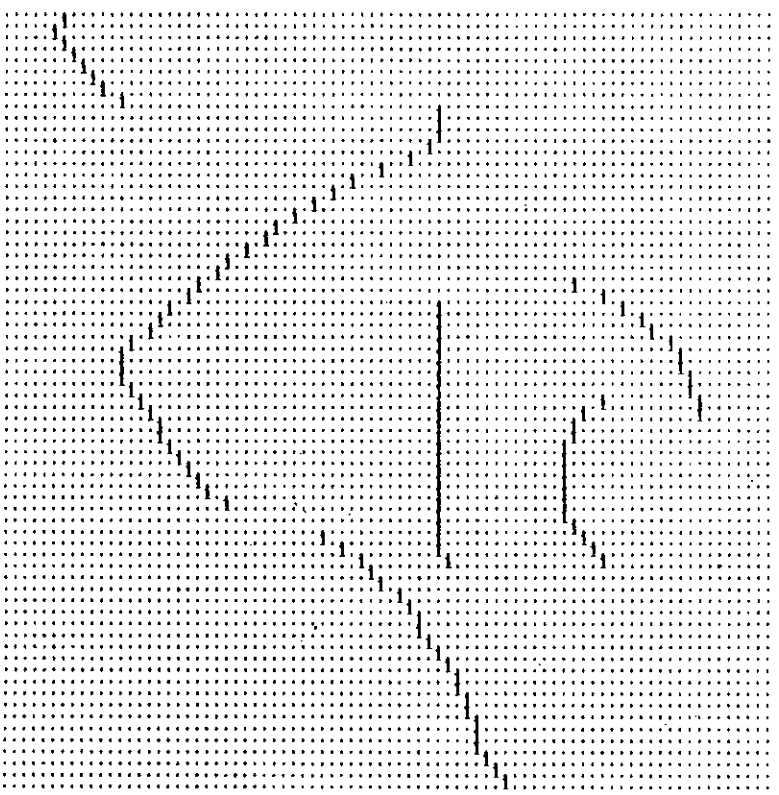


Fig. 3.8 Character क, after filtering out stray points.

Filtering out is performed by and-ing each byte with the mask of its top and bottom

bytes and or-ing these two results. The result is stored in the byte location. For the first and last line only the bottom and top byte is taken respectively. After filtering out all stray points the contour of Fig. 3.6 becomes as shown in Fig. 3.8. Procedure FilterOut shows the filter out process. [For source coding refer to *Filterout()* function in Appendix C].

```
function FilterOut;
  begin
    if y=1 then  {first line}
      filter out an isolated  {an '1'is considered as to be
       '1' comparing with      isolated if in the next line or
        the second line;       in the previous line there is no
                               '1' within three bit left or three
                               bit right}
    for y:=2 to LetterHeight-1 do
      begin
        filter out an isolated '1' comparing with
          the previous line or next line;
      end;
    if y=LastLine then  {last line}
      filter out an isolated '1' comparing with
        the previous line;
  end;
```

3.5 EXTRACTION OF STROKES (MORPHS).

After getting a clean contour of the letter, next task is to extract the strokes. All the x coordinates, and only the starting and ending y coordinates of the strokes are stored in a memory location.

Storing of stroke coordinates is done by scanning the letter contour from left to right and from top to bottom starting from the upper left corner. Whenever a black

point is found, scanning proceeds to the bottom points for finding continued points. Points are assumed continuous when a point is within three bits left or three bits right of its immediate upper line black. If a discontinuity occurs, scanning proceeds to the next line with the assumption that the void in the current line is due to improper scanning. If the next line gives continuity, a black point is assumed in the previous line and its coordinate is stored. Otherwise, the stroke is assumed to have terminated and its ending y coordinate is stored. All black points of the stroke are nulled out to '0', during the time of x coordinate storing so that these are not considered again when scanning starts from the upper left corner for the next stroke. The process of scanning strokes continues until no new strokes can be found. The result is to produce a collection of stroke coordinates. Algorithm StoreStrokes is used to extract the strokes of the character.

```
function StoreStrokes;
  begin
    allocate memory for storing the stroke coordinates;
    repeat
      for y:=1 to LetterHeight do     {scan all the bytes starting
        begin                          from top left corner for
          for x:=1 to LetterWidth do   getting a 'black' byte}
          if the byte is 'black' then {a byte is 'black' if it
                                       contains black(1) dots}
```

```
          goto StartFound;
        else
          proceed to next byte;
      end;

    StartFound:
      save starting y coordinate;
      save starting x coordinate;
      while y<=LetterHeight do  {find out the remaining
        begin                    points of the stroke}
          if a continued black bit is found in the next line
          then
            begin
              save x coordinate;
              zero out the black found;
            end;
          else if continued black bit is found in the second
                  next line
            then
              begin
              save x coordinates;
              zero out the black found;
            end;
          else  {means an end of stroke}
            save end y coordinate;
        end;
        update variables for next stroke;
  until all black points have been processed;
end;
```

3.6 ENCODING OF STROKES (MORPHS).

For the purpose of recognition, however, it is desired to generate a numeric code for each character. The first step in this procedure is the selection of morphs (or strokes) and the generation of code for each stroke(component) in a character.

3.6.1 Selection of Strokes.

In syntactic method each input pattern is to be resolved into primitive structural elements, called *strokes(morphs)*. The first step in designing a syntactic model is the selection of morphs in terms of which the patterns of interest can be represented. In selecting morphs care must be taken so that they are simple to recognize and they are minimum in number. For the present analysis, six simple strokes are chosen to represent all 50 Bangla characters, which are shown in Table 3.1. A wide range of deviation is allowed to each of the six morphs. Fig. 3.9 shows a diagrammatic representation of lines and their deviation limits.

3.6.2 Finding Stroke Code.

The coordinates for a particular stroke are now analyzed to find the code of that stroke. The stroke under consideration is either a curve or a straight line. For recognizing a curve, regression analysis is performed to find the co-efficients of the equation

$$x = a + by + cy^2$$

If N is the total number of points in the stroke, the value of $cN(= c*N)$ is evaluated. If absolute value of cN is greater than 0.75, the stroke under consideration is a curve, otherwise, it is a straight line. The limiting value, 0.75 of cN allows a wide range

TABLE 3.1

STROKES USED FOR RECOGNIZING BANGLA CHARACTERS

Stroke	Characteristics	Numeric Code
	Horizontal line	1
	Positive sloping line	2
	Vertical line	3
	Negative sloping line	4
	Vertical concave curve	5
	Vertical convex curve	6
	Rejected stroke	0

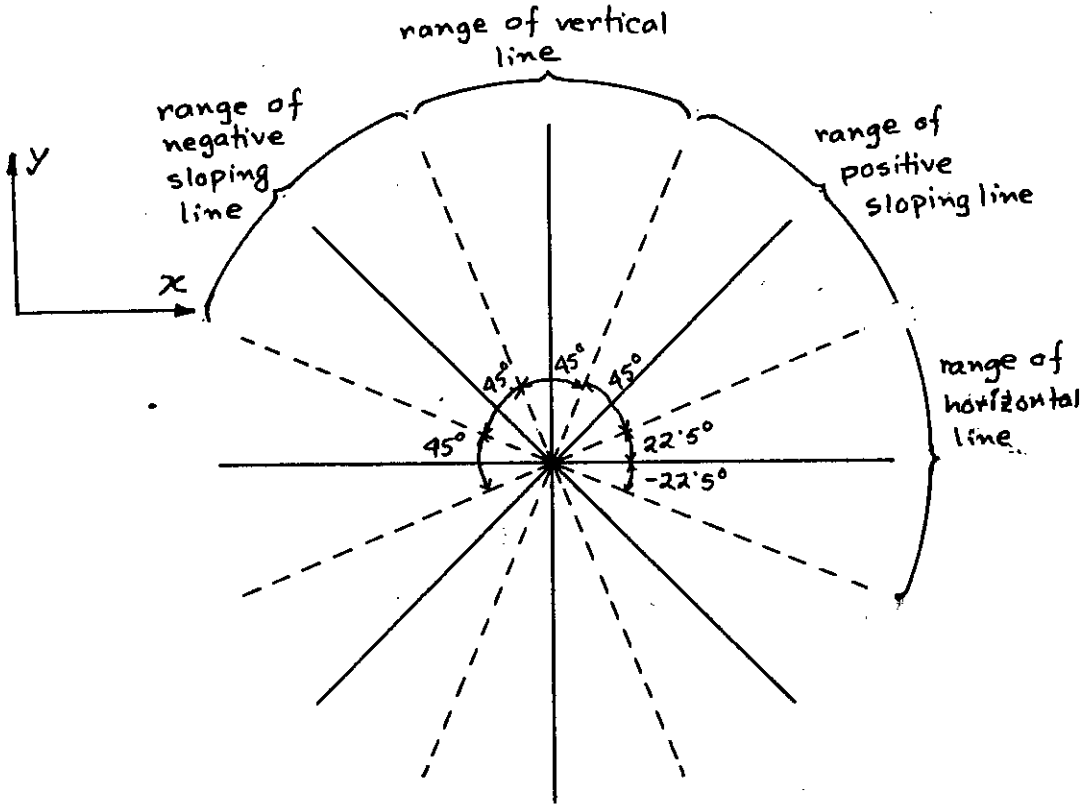


Fig. 3.9 Ranges for different straight lines.

of flexibility.

When a stroke is a curve, it may be a vertical concave curve or a vertical convex curve. If the value of c is positive, the stroke under consideration is a vertical concave curve, otherwise, it is a vertical convex curve. For separating straight lines, regression analysis is performed again to find the co-efficients of the equation

$$x = a + by.$$

Depending on the value of the slope of the line (reciprocal of b), a decision is taken whether it is a horizontal line, a vertical line, a positive slope line, or a negative slope line.

The objective of a recognition system is to maximize the probability of correct

85057

classification of each character. To minimize the probability of misclassification, a noise level is chosen by observing the structure of the characters and applying the intuition. If the stroke length (N) is less than three, or less then 6.5% of the letter height, or less than 6.5% of the letter width, a value of '0' is assigned to the stroke code. It is assumed that a stroke with stroke code '0' was created due to noise present in the surrounding of the original letter or for some other reasons. This type of stroke is rejected and the scanning continues.

An Algorithm for finding the stroke code of a particular stroke is described formally in procedure FindStrokeCode. [For source coding refer to *FindStrokeCode()* function in Appendix C].

```
function FindStrokeCode;
  begin
    N:=StrokeLength;
    if N<4 and N<6.5% of LetterHeight and N<6.5% of LetterWidth
      then
        begin
          StrokeCode:='0';     { rejected code }
          return;
        end;
    calculate coefficient c of the equation x=a+by+cy2 by
    regression analysis;
    if abs(c*N) < 0.75 then  {the stroke is a straight line}
      begin
        calculate Slope of the line x=a+by by regression analysis
        if Slope>= -0.4142 and Slope<=0.4142 then
```

```
        begin
          StrokeCode:='1';  { Horizontal line }
          return;
        end;
      if Slope> 0.4142 and Slope<=2.4142 then
        begin
          StrokeCode:='2';  { positive sloping line }
          return;
        end;
      if Slope> 2.4142 or Slope<=-2.4142 then
        begin
          StrokeCode:='3';  { vertical line }
          return;
        end;
      if Slope>=-2.4142 or Slope< -0.4142 then
        begin
          StrokeCode:='4';  { negative sloping line }
          return;
        end;
    end;
  else   {the stroke is a curve}
    if c > 0 then
      begin
        StrokeCode:='5';    { vertical concave curve }
        return;
      end;
    else
      begin
        StrokeCode:='6';    { vertical convex curve }
        return;
      end;
end;
```

3.7 ENCODING OF CHARACTERS.

The representation of characters is in the form of a numeric string. The numerals in the code not only indicate the morphs which build the character but also show the relationship among the morphs. The strokes in the character are visited from left to right and from top to bottom along various rows and columns. All strokes are traversed exactly once. The letter code consists of the stroke codes written down in the order in which they are encountered. It is possible to get an idea about the structure of the character from its code. For example character *Kaw* gives a code 556354 which indicates a structure as depicted in Fig. 3.8.

Procedure GetLetterCode shows an algorithm for finding the letter code of a particular character. [For source coding refer to *GetLetterCode()* function in Appendix C].

```
function GetLetterCode;
  begin
    for i:=1 to TotalStroke do
      begin
        point to stroke under consideration;
        FindStrokeCode;
        if StrokeCode != '0' then  {StrokeCode '0' means
                                    rejected stroke.}
          update LetterCode;
      end;
  end;
```

3.8 RECOGNITION OF CHARACTERS.

The characters represented by numeric string codes are now to be recognized. A dictionary of codes is searched to find a matching code corresponding to the code generated from the input character. The dictionary is constructed giving codes corresponding to each acceptable appearance of a character. The codes are acquired from five sets of characters having variable size and obtained from different sources. A sample of these sets appear in Fig. 3.10. Initially, all 50 characters of 5 different sizes are processed 5 times each so as to provide a dictionary for later tests. Table 3.2 shows the results of initial test.

The results show a reasonable structural representation produced for all the characters. Form table 3.2, it is evident that some of the characters can be represented uniquely as there are natural differences in structural compositions among the characters. Most of the characters have got more than one code depending on the printing styles, sizes and quality of the digitized output. In a few cases, the same code represents a pair of characters having structural similarities. Elaborate discussions about characters having same code and characters having more than one code are presented in the subsequent sections.

3.8.1 Characters Having More Than One Code.

Almost all of the characters have several codes due to the variation of the output of the scanner and the variation of the input pattern. One point to note here is that to each character deviation to such an extent is always allowed as is allowed to each stroke. The dictionary has the entry for all the possible combination

অ	আ	ই	ঈ	
		উ	ঊ	ঋ
	এ	ঐ		
	ও	ঔ		

ক	খ	গ	ঘ	ঙ
চ	ছ	জ	ঝ	ঞ
ট	ঠ	ড	ঢ	ণ
ত	থ	দ	ধ	ন
প	ফ	ব	ভ	ম
	য	র	ল	
	শ	ষ	স	হ
	ড়	ঢ়	য়	ৎ
	ং	ঃ	ঁ	

Fig. 3.10 (a) Bangla characters of sample # 1

স্বর বর্ণ

অ আ ই ঈ উ
ঊ ঋ এ ঐ ও ঔ

ব্যঞ্জন বর্ণ

ক খ গ ঘ ঙ
চ ছ জ ঝ ঞ
ট ঠ ড ঢ ণ
ত থ দ ধ ন
প ফ ব ভ ম
য র ল শ ষ
স হ ড় ঢ় য়
ৎ ং ঃ ঁ

Fig. 3.10(b) Bangla characters of sample #2

অ আ ই ঈ

উ ঊ ঋ এ

ঐ ও ঔ

ক	খ	গ	ঘ	ঙ
চ	ছ	জ	ঝ	ঞ
ট	ঠ	ড	ঢ	ণ
ত	থ	দ	ধ	ন
প	ফ	ব	ভ	ম
য	র	ল	শ	ষ

Fig. 3.10(c) Some Bangla characters of sample #3.

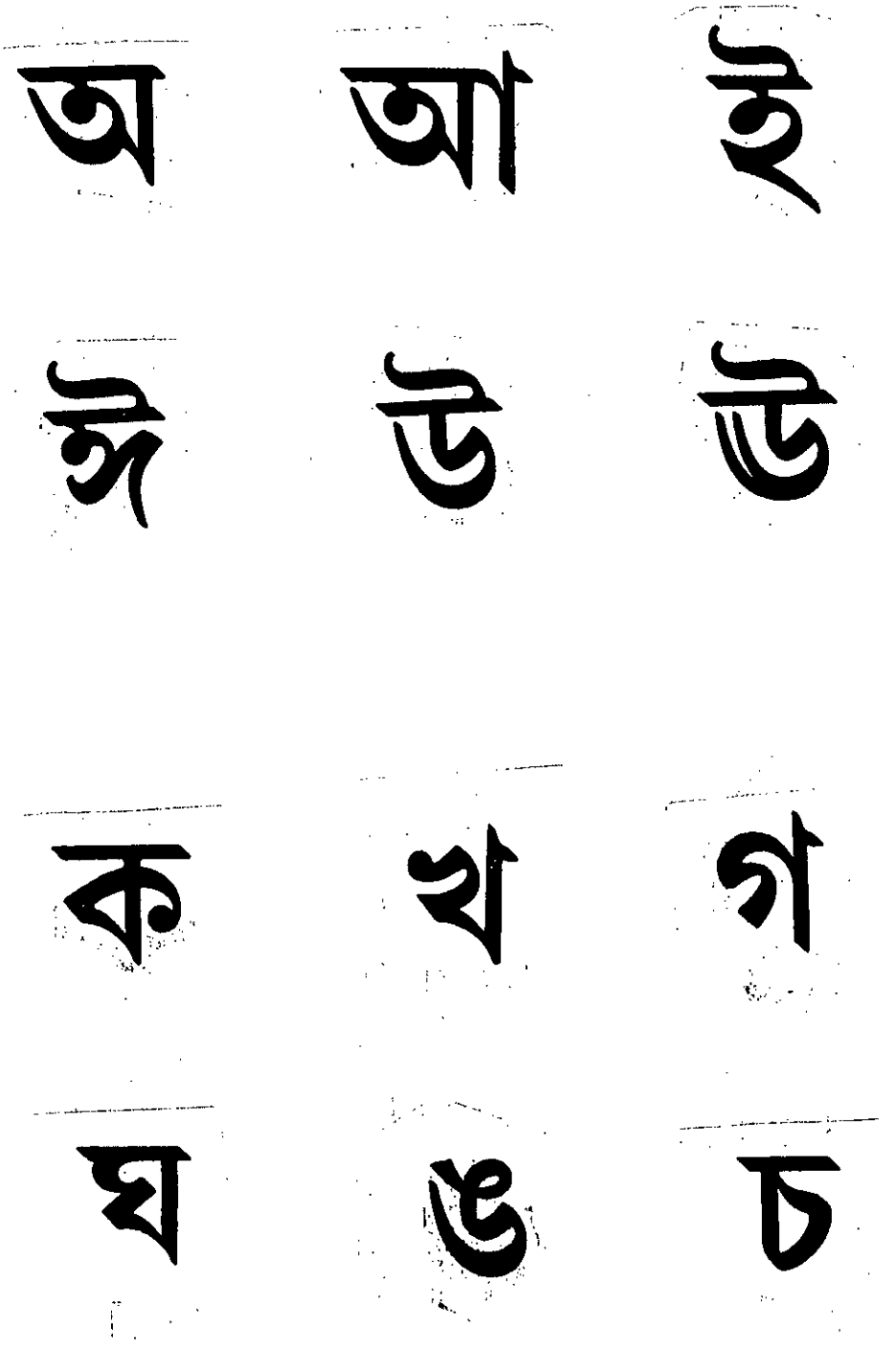


Fig. 3.10(d) Some Bangla characters of sample #4.

Fig. 3.10(e) Some Bangla characters of sample No.5.

TABLE 3.2

SUMMARY OF CODES OBTAINED BY RUNNING THE CODING PROGRAM ON FIVE SAMPLE SETS OF CHARACTERS.

letter name	sample #1	sample #2	sample #3	sample #4	sample #5
অ	435464 535464	453464	453464 435464 534564 434564	435464	434564
আ	34533464 34335464	35533464 34353464	35533464 34533464 34365464 3453464	35335464 34533464 34335464	34334564
ই	5355644	5645644 564565	564565 5645644 5645646 534564	5645646	5645644
ঈ	56432633 56432635	56552635 56422634	56422635 56452635 56552634 56452645 56452634	56452634 56552634 56552635	56452645
উ	5653326 5643326	5653426 5643426	5643426	5653426 5643426	5643426
ঊ	56434326 56433426	56533426 46434326	56434326 56433426	56434326 56433426 56534326	56433426
ঋ	33255344	33255346	33255346 33255344	33255364 33525344 33255344	33655346
ঌ	5354	5354 5344	5354 5344	5354 5356	5354
ঐ	565334 565354	565354	565354	565354	565354
ও	56456	56456	56456	56456	56456
ঔ	5656456 5654356	5656456	5656456	5656456 5654656	5656456
ক	4656356	4653654 5653456 5653654 553454	4653656 5653456 5656356 453654	453656 4653656 456354 456356	4656354

Continuation of TABLE 3.2

letter name	sample #1	sample #2	sample #3	sample #4	sample #5
খ	325644 325646	325644 325646	325646	325644 325646	325644 325646
গ	3535	3535	3535	3535	3535
ঘ	4354	4356 4354	4356 4354	4354	4354
ঙ	546326	426526 426326 246426 546326	426426 246466 546426	526426	546426
চ	536 436	556 536 436	536 436	436	436
ছ	436646	436644 436646	436646	436644 436646	436644
জ	455336 4554326	454436 555436 554466 554646 554436 4554326	554446 4545646 4544263 5544246 4554236	453436 4555636 4535263 4545236	5545366 5545636
ঝ	345344 355344	345344 355346	345344 345364	345344	345344
ঞ	523655644 253655644	523655344 523655654 553645654 553645656	523655644 523645644	553655654 553655644	535565644
ট	56533 565352	56434 564356	56534 56434 564356	564352 564356	564352
ঠ	3436	5426 5456 3426	5426 3426	3456 5456 3426 5436	5456
ড	53426 43426	53426	43426	43426	43426
ঢ	533 433	5352 5356 4352 534 4552 4356	4356 4352	4356 433	4352

Continuation of TABLE 3.2

letter name	sample #1	sample #2	sample #3	sample #4	sample #5
ণ	3533 3535	3533 3535	3535	3535	3535
ত	4456	4456 5456	4456	4456	4456
থ	35646 35644	35646	35646	35646	35644
দ	4323	4323 323	4323	4323	4323
ধ	36434 36436	565534	565534 564534 655536	565536 564534 653534	564536
ন	4354 5354	4354	4354 5354	4354 5354	4354
প	35633 35233	35623 35232	35632 35232	35632 35232 35623 35633	35632
ফ	435656	435656 435654	435656 435354 435356	435656 435654	435656
ব	46536	46534	46534 4534	46536 4536	46536 4536
ভ	44526	44526 54526	44526	44526 54526	44526
ম	53654 43654	43654	43654	43654	43654
য	4356 4354	4356 4354	4354	4356	4356 4354
র	55345 465345 565345	465365 465345 565365	465365 465345	565345 465345	465365 465345
ল	4325345	4352435 5355345 4355365	4352345 5325345 4355365 4355635	4355435 4355345 4325435	4355635 4355345
শ	352335 325334	352235 342234	342235 342534 345234	345535 342535 355535 345335 324334 342335	342235 345235

Continuation of TABLE 3.2

letter name	sample #1	sample #2	sample #3	sample #4	sample #5
ঝ	43536	43536 43534	43536 43534	43536 43534	43534
ঞ	43635	43634	43635	43635 43653	43635
ঠ	465634 456444 45644 556464	45644	45646	465646 465644 4565 4564	45644
ড়	433265	434265	434265	434265 433265	434265
ঢ়	43565	43565 53565	43565 4365	43565 43525 4335	43525
য়	43565 43545	43565 53545	43565 43545	43545	43545
ৎ	5636 5654	5656 563 564	5656 5646 5644	5656 5636 565 563	5656 565
ৎ	565	565	565	565	565
০	5656	5656 5256 5652	5656	5656	5656
ৎ	534 565 564	542	425 453 552 435	452 542 453	542

of the codes of a character.

Fig 3.11(a) shows two different structures of character ব which are scanned from the same source. The first ব resolves into a positive sloping line, a vertical convex curve, a vertical concave curve, a vertical line and another positive sloping line in sequence, giving rise to a code 46534. For the second ব, the first four strokes are identical to those of the first letter but the fifth stroke is a vertical convex curve and consequently gives a code 46536.

Fig 3.11(b) shows four different structures of character ল which are scanned from different sources. The first one with code 4352345 has the first stroke with negative sloping line, while the others has this stroke with vertical concave curve. This variation is only due to the difference in the output of the scanner. According to the scanned image the third and fourth stroke, and also the fifth and sixth stroke may interchange their positions which are depicted by the first (code 4352345) and fourth (code 5325435) ল, and second (code 5355345) and third (code 5355435) ল.

3.8.2 Characters With Same Code.

There are several groups of characters which show same code due to structural similarity among them. All the characters having same code are grouped together. List of such grouping is given in Table 3.3. The characters of the same group are distinguished by another test. This test is performed by comparing the stroke ratio of a specific stroke for different members of the same group. The *stroke ratio* of a particular stroke is defined as the ratio of the length of that stroke (number of points on the stroke) to the length of the letter (the height of the letter in number

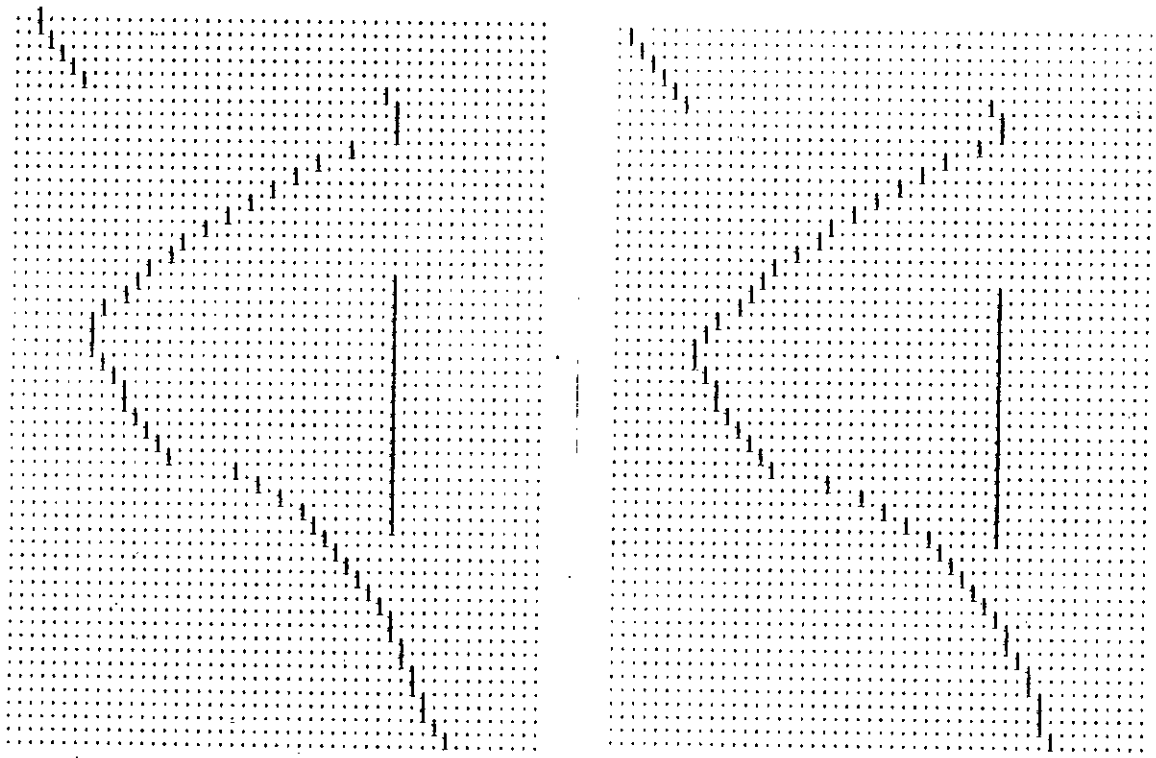


Fig. 3.11(a) Two different structures of character ঘ.

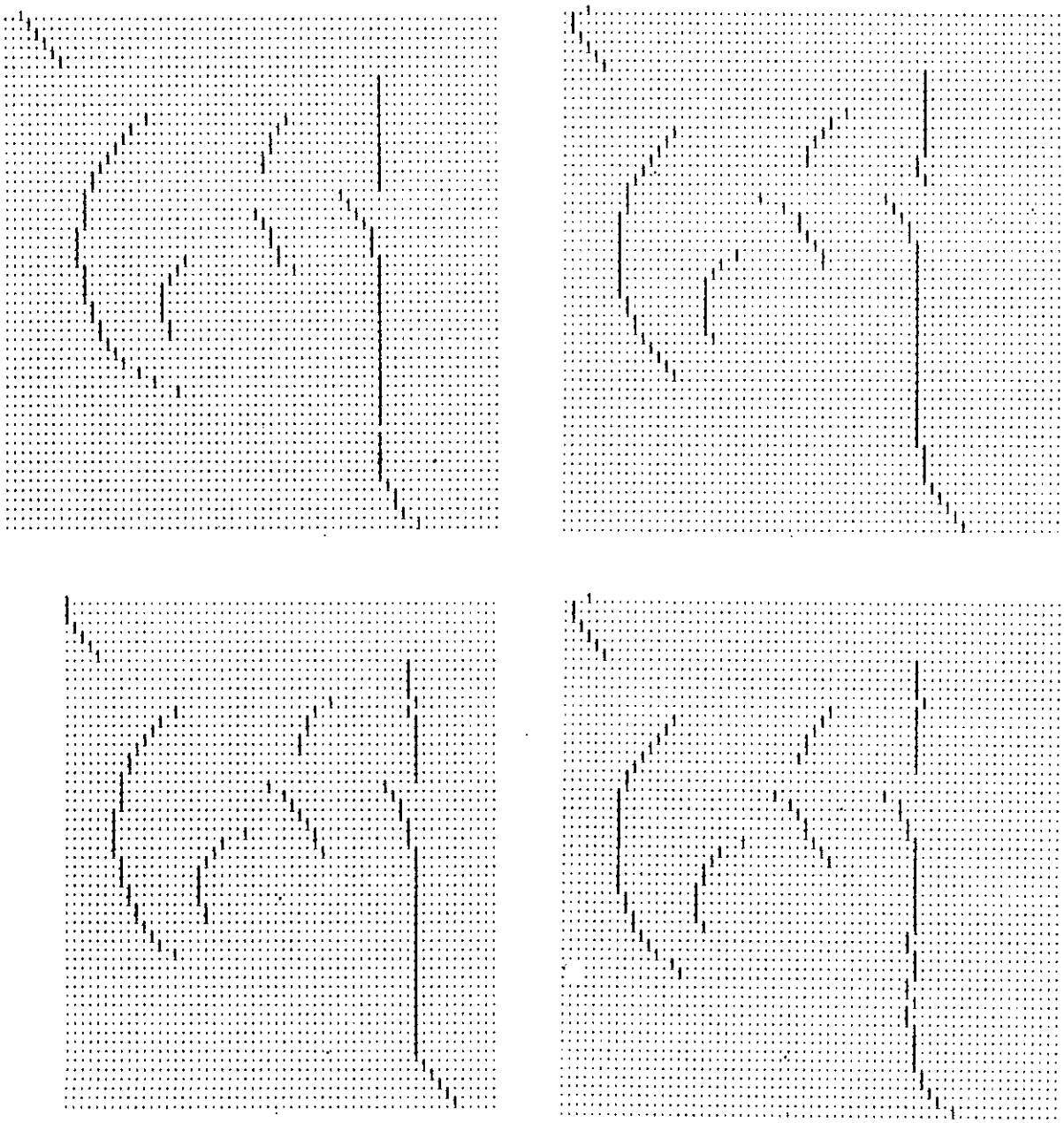


Fig. 3.11(b) Four different structures of character ल.

of pixels). Third column of the table 3.3 shows the test based on the stroke ratio for all the character groups having same code.

Characters ঘ ,য ,ন : These characters give rise to code 4354 due to structural resemblance among them. The bit maps of the contour of these characters are shown in Fig. 3.12(a). For ঘ , the stroke ratio of the first stroke is always greater than 27%. By testing the first stroke ratio, ঘ can be isolated from য and ন . For ন , the stroke ratio of the second stroke is always less than 50%. By this second test ন can be isolated from য . Hence, if the first stroke ratio is greater than 27%, the character will be recognized as ঘ , and if the first stroke ratio is less than 27% and second stroke ratio is greater than 50%, the character will be recognized as য , otherwise, the character is ন .

Characters গ, ণ: Characters গ and ণ have got striking structural similarities. Both give rise to code 3535 which means each of গ and ণ is made up of four strokes, two vertical concave curves and two vertical lines. The bit maps of the contour of the two characters are shown in Fig. 3.12(b). For distinguishing গ and ণ , another check is employed which is based on the structural difference. For গ , the stroke ratio of the fourth stroke is always greater than 25%. If the fourth stroke ratio is greater than 25%, the character is গ , otherwise the character is ণ .

TABLE 3.3

GROUPS OF CHARACTERS HAVING SAME CODE DUE TO STRUCTURAL SIMILARITY AMONG THE MEMBERS OF A GROUP.

character code	character name	test
3535	গ	$S_4 > 0.25$
	ন	$S_4 < 0.25$
433/453/ 533/553	ত	$S_1 < 0.20$
	ঙ	$S_1 > 0.20$
5354	ন	$S_1 > 0.20$
	ন	$S_1 < 0.20$
5356	ড	$S_1 > 0.20$
	ট	$S_1 < 0.20$
5456	ত্ত	$S_1 > 0.20$
	ত	$S_1 < 0.20$
563/564	ঙ	$S_2 < 0.20$
	ঙ	$S_2 > 0.20$
565	ং	$S_3 > 0.45$
	ং	$S_3 < 0.45$
5656	ং	$S_1 > 0.55$
	ং	$S_1 < 0.55$
5656356/ 5656456/ 5656656	ঙ্ক	$S_1 > 0.17$
	ক	$S_1 < 0.17$
4354	ঘ	$S_1 > 0.27$
	ন	$S_1 < 0.20$ & $S_2 < 0.50$
	ঘ	$S_1 < 0.27$ & $S_2 > 0.50$
4356	ঘ	$S_1 > 0.27$
	ঢ	$S_2 > 0.75$
	ঘ	$S_1 < 0.27$ & $S_2 < 0.75$

where S_i stands for Stroke Ratio for the ith stroke.

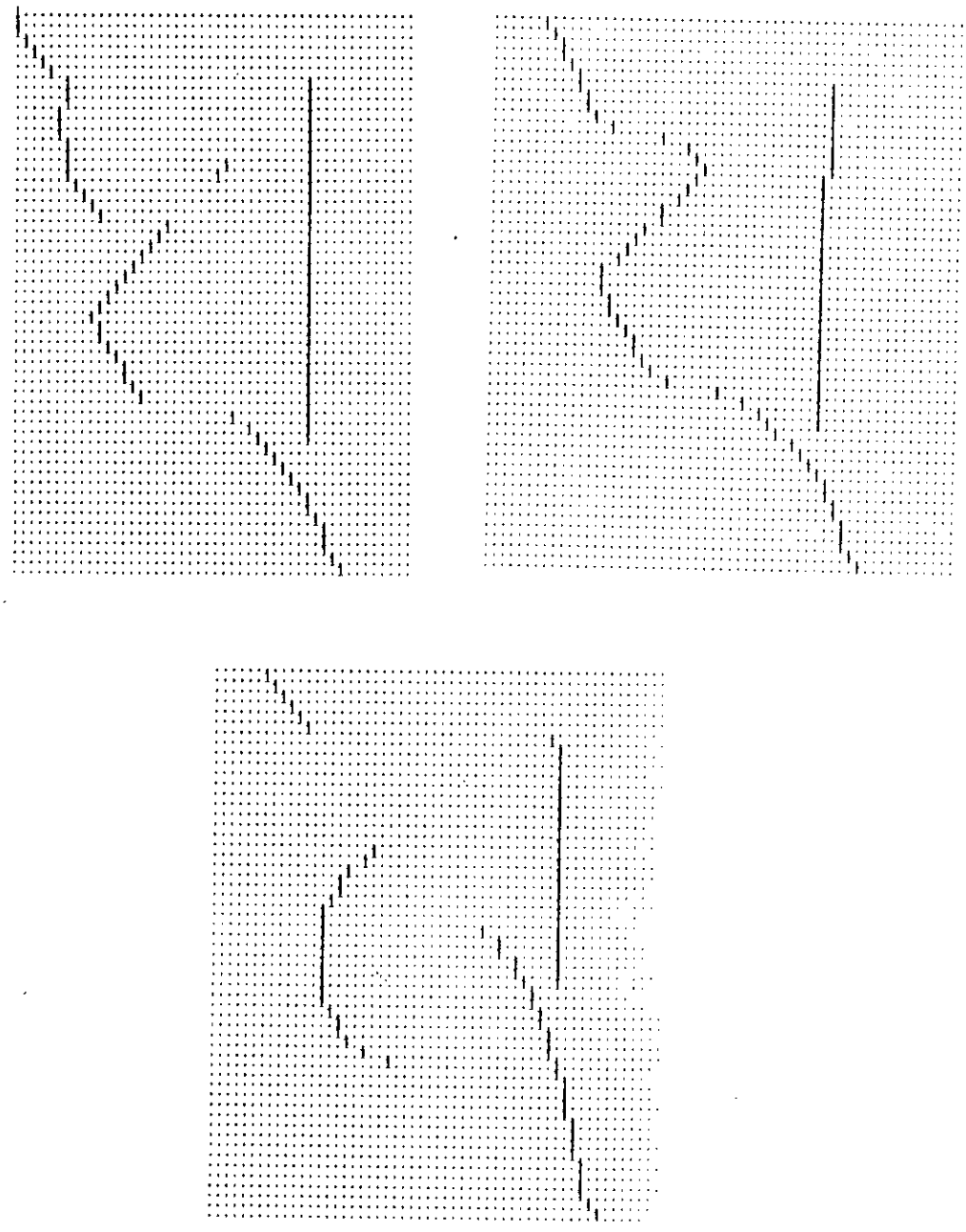


Fig. 3.12(a) Bit maps of the contour of characters ঘ, ঘ, ন,

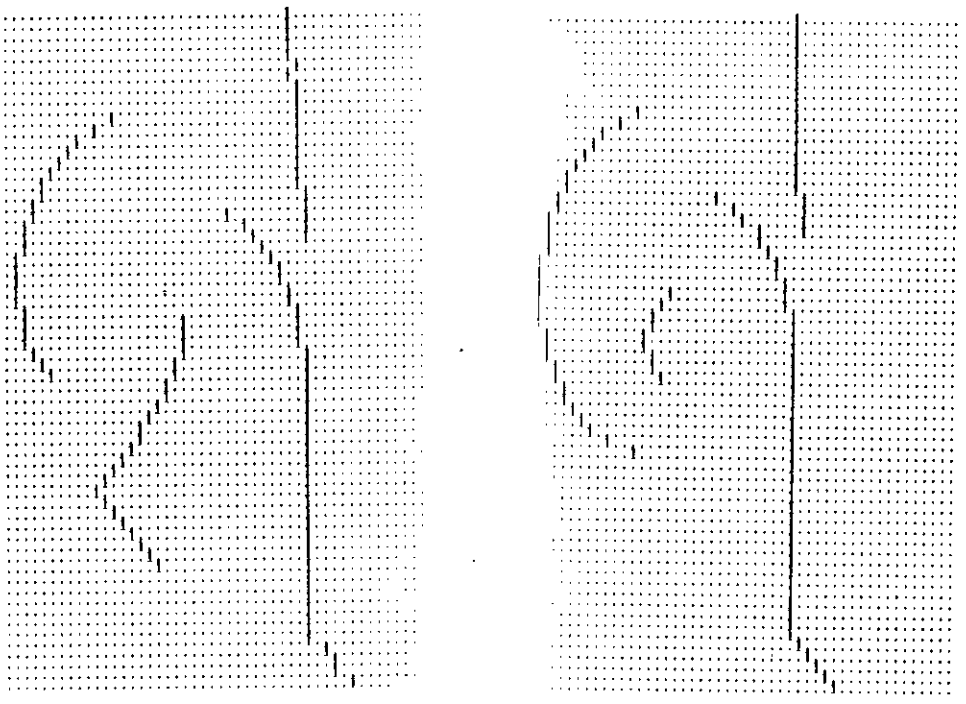


Fig. 3.12(b) Bit maps of the contour of characters ſ, T.

3.8.3 Development of a Dictionary.

A dictionary is developed giving codes corresponding to each acceptable appearance of a character (Appendix B). For the ease of searching, the total volume of the dictionary is first divided into five subgroups (since there is no letter code having the stroke code '1'):

Group '2' : Character code starting with stroke code '2'.

Group '3' : Character code starting with stroke code '3'.

Group '4' : Character code starting with stroke code '4'.

Group '5' : Character code starting with stroke code '5'.

Group '6' : Character code starting with stroke code '6'.

Depending on the value of the starting stroke code, the program switches to any one of the above groups. The algorithm is similar to m-way search tree (MWT). Each of the above subgroups is again divided into more than one subsubgroups depending on the value of the second stroke code number. Depending on the value of the third stroke code, each subsubgroup is divided into more additional groups and so on.

CHAPTER FOUR

DISCUSSIONS, CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

No, no: I never guess. It is a shocking habit — destructive to the logical faculty. what seems strange to you is only so because you do not follow my train of thought or observe the small facts upon which large inferences may depend.

Sherlock Holmes

CONTENT

CHAPTER 4. DISCUSSIONS, CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

4.1 General Remarks 85

4.2 Results 86

4.3 Conclusions 88

4.4 Suggestion for Future Work 90

CHAPTER 4

DISCUSSIONS, CONCLUSION AND SUGGESTIONS FOR FUTURE WORK

4.1 GENERAL REMARKS.

Syntactic pattern recognition seems to be a promising tool for the recognition of Bangla characters. In this very initial stage of Bangla character recognition, perhaps the most important area of investigation would be to deal with printed Bangla characters. An effective Bangla printed character recognition system would permit the rapid processing of vast amounts of printed materials in this language.

Recognition of characters is a very involved task as characters of allowable printing sizes and styles as well as variations in quality of the digitizer output have to taken into consideration. A program has been written implementing the algorithms developed after studying the digitized character structures of different sizes of each character. The program has been tested with a number of characters from several different sources. The tests were designed to consider three aspects:

a) To see whether the program develops consistent codes for characters of the same font, i.e., whether the two instances of the same character from the same source yield the same class of codes.

b) To see whether the program works for characters from different sources of same size.

c) To see whether factors such as character size and character complexity affect program performance.

4.2 RESULTS.

A test is carried out using five sets of characters which are different from the initial sample of characters. Two instances of each character from the same source is processed for a total of 500 new characters. To see whether the program developed gives consistent code for characters of same font i.e., whether the program can recognize the characters properly a test run on the above mentioned 500 samples is performed. The result of the test is shown in Table 4.1. Out of total 500 samples, the program recognizes 467 times correctly, 11 times incorrectly. In 22 cases it fails to give a decision i.e., it rejects the input pattern. So, from Table 4.1, it is observed that test (a) gives 93.4 % accuracy and test (b) gives 100% accuracy within allowable printing styles. In test (c) it has been found that some ambiguity may arise if a stroke in a character becomes so small as to be comparable to the width or height of the character. That is, to a large extent the program is size invariant. Problems may occur only when some of the strokes in a character are so small as to be of the same order of magnitude as what is considered noise for others.

TABLE 4.1

SUMMARY OF THE RESULT OBTAINED BY RUNNING THE RECOGNITION PROGRAM ON A TOTAL OF 500 SAMPLES.

character name	# of samples	properly classified	wrongly classified	rejected
অ	10	10	-	-
আ	10	10	-	-
ই	10	9	-	1
ঈ	10	9	-	1
উ	10	10	-	-
ঊ	10	10	-	-
ঋ	10	9	-	1
এ	10	10	-	-
ঐ	10	10	-	-
ও	10	10	-	-
ঔ	10	10	-	-
ক	10	10	-	-
খ	10	10	-	-
গ	10	10	-	-
ঘ	10	9	-	1
ঙ	10	9	-	1
চ	10	8	1	1
ছ	10	9	-	1
জ	10	7	1	2
ঝ	10	10	-	-
ঞ	10	9	-	1
ট	10	8	1	1
ঠ	10	8	-	2
ড	10	10	-	-
ঢ	10	9	-	1
ণ	10	10	-	-
ত	10	9	1	-
থ	10	10	-	-
দ	10	10	-	-
ধ	10	9	-	1
ন	10	8	1	1
প	10	9	-	1
ফ	10	8	1	1
ব	10	9	1	-
ভ	10	10	-	-
ম	10	9	1	-
য	10	9	-	1
র	10	10	-	-
ল	10	9	-	1
শ	10	10	-	-
ষ	10	9	1	-
স	10	10	-	-
হ	10	10	-	-
ড়	10	10	-	-
ঢ়	10	8	1	1
য়	10	10	-	-
ৎ	10	10	-	-
ং	10	9	1	-
ঃ	10	10	-	-
ঁ	10	9	-	1
	500	467	11	22

The failures are due to a particular component not being analyzed, improper output of the scanner, stroke segments appearing at different angles and strokes touching in one case but not the other. Most of the problems has occurred in the *StoreStrokes* routine, which is supposed to isolate a stroke and store its coordinates. The *StoreStrokes* routine would sometimes make mistakes if, for example, two strokes were very close together or one stroke was divided into two parts by more than three lines. Another source of failure may be due to unavoidable noise present in the character.

4.3 CONCLUSIONS.

For the present analysis, the basic idea of syntactic method — to represent each complex pattern in terms of simple subpatterns, is utilized. But the techniques developed for resolving each character into a number of strokes, their recognition lead to a new approach. However, the idea of representation of characters by numeric codes for their recognition was proposed and successfully used before in recognizing foreign letters, but again this is the first time for Bangla characters.

The technique developed for selecting strokes will no doubt play a significant role in the field of pattern recognition. Regression analysis have been used for recognizing the strokes. The present technique for recognizing strokes is quite simple and efficient. Further improvement can be suggested so as to reduce the stroke recognition time.

The character codes of the present analysis are quite informative. The code gives information about the strokes building up the character as well as their relation.

The principal difficulty with this approach is the lack of use of a nearness criterion. If a character sample produces a code different from all known codes, it is rejected. As a partial solution to this problem, some of the characters have been assigned several codes, corresponding to common discrepancies between character samples due to a lack of uniformity in print quality. Improvements in resolution would also be expected to help.

Bangla text recognition is made difficult by the size of the character set and the complexity of the individual characters. Test results indicate that use of the approach suggested by the present work would necessitate a dictionary in which some characters are associated with several codes.

Several possibilities exist which could improve the chances of constructing a practical text recognition device.

a) **High standard of Print Quality:** A device restricted to use only with very high-quality print should be more consistent in code generation, thus reducing the size of the required dictionary.

b) **Stylized Font:** A specially designed font tailored to the recognition algorithm would improve the algorithm's performance.

c) **Language Simplification:** Steps to reduce the number of composite characters in general use and the complexity of individual composite character would

simplify text recognition program.

The results reported here lead the author to believe that pattern analysis can be a fruitful approach to Bangla character recognition. Like every method the present one naturally has got some limitations as mentioned; but still the performance of the present method in recognizing Bangla alphabetic characters has found to be quite satisfactory. Also the technique developed for the present analysis can be successfully implemented to other fields of pattern recognition.

4.4 SUGGESTION FOR FUTURE WORK.

Further research work in automatic recognition of Bangla character should concentrate on

a) Developing an algorithm for building a dictionary which also includes composite characters.

b) Developing an efficient algorithm for separating the characters from a text.

c) Investigation to find if statistical measurement for recognizing strokes result in a more efficient technique.

d) Modification of the present dictionary searching algorithm for implementing in composite character recognition.

Considerable work can be done in the area of automatic translation of Bangla text. A Bangla reading and translating machine, which could scan printed Bangla and produce English output, would provide the most desirable means of improve-

ment. Such a machine is a long way down the road, but individual steps which advance toward that goal are to be encouraged. One answer to the problem would be the development of a practical Bangla character recognition machine, toward which the effort of this project is directed. It is hoped that advances in this area would provide additional incentive for work in translation devices. On a more modest scale, a Bangla character recognition device could be used as a type of on-line dictionary to speed up the process of human translation.

Somewhat more difficult than the recognition of printed Bangla characters is the recognition of handwritten Bangla characters. Not only is the variability of writing style in a given individual greater than that for a single font of printed characters, but differences in writing style from individual to individual are greater and less predictable than font-to-font differences. An on-line handwritten Bangla character recognition device would also be a step in the right direction. Such a device, again, could be used for on-line dictionary look-up and would also provide a rapid means for written Bangla transcription.

REFERENCES

REFERENCES

1. R. C. Gonzalez and M.G. Thomason: *Syntactic Pattern Recognition, An Introduction*: 1982, Addision - Wesley Publishing Company, Reading, Massachusetts, U. S. A.
2. D. Dutta Majumder: *Pattern Recognition Methods and Applications*, Recent Developments in Pattern Recognition and Digital Techniques: pp. 150 - 190, Indian Statistical Institute, 1977.
3. Md. Abdus Sattar: *Recognition of Printed Bengali Characters by Decision-Theoretic Method of Pattern Recognition*: M. Sc. Engg. Thesis, 1990, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.
4. K. S. Fu: *Digital Pattern Recognition*: ed. by K. S. Fu, pp. 1 - 14, Springer-Verlag, New York, 1976.
5. Lavven Kanal: *Patterns in Pattern Recognition*, 1968 - 1974: IEE Trans. Information Theory, vol. IT - 20, pp. 697 - 722, Nov. 1974.
6. Kendall Preston Jr.: *A Comparison of Analog and Digital Techniques for Pattern Recognition*: Proc. IEEE, vol. 60, pp. 1216 - 1231, Oct. 1972.

7. S. H. Unger: *Pattern Detection and Recognition*: Proc. IRE. vol. 47, pp. 1737 - 1752, October 1959.

8. L. P. Hoewitz and G. L. Shelton: *Pattern Recognition Using Auto-correlation*: Proc. IRE. January 1961, pp. 175-185.

9. P. G. Perotoo: *A New Method for Automatic Character Recognition*: IEEE Trans. on Electronic Computers, pp. 521 - 526, October 1963.

10. H. A. Glucksman: *A Parapropagation Pattern Classifier*: IEEE Trans. on Electronic Computers, pp. 434 - 443, June 1965.

11. S. S. Yau and C. C. Yang: *Pattern Recognition by Using Associative Memory*: IEEE Trans. on Electronic Computers, pp. 944 - 947, December, 1966.

12. A. W. Holt: *Comparative Religion in Character Recognition Machines*: IEEE Computer Group News, vol. 2, pp. 3 - 11, Nov. 1968.

13. K. S. Fu: *Introduction to Digital Pattern Recognition*: ed. by K. S. Fu, pp. 95 - 134, Springer-Verlag, New York, 1976.

14. M. A. Sattar and S. M. Rahaman: *An Experimental Investigation on Bangla Character Recognition System*: Bangla Computer Soceity Journal, vol. - 4, no. 1, Dec. 89, pp. 1 - 4.

15. W. Stallings: *Computer Analysis of Printed Chinese Characters*: M.I.T. Cambridge: Ph.D. Thesis (1971).

16. Y. Nakano, K. Nakata, Y. Uchihura, A. Nakajimo: *Improvement of Chinese Character Rrecognition Using Projection Profiles*: Proceedings of the First International Joint Conference on Pattern Recognition, Washington (1973).

17. Md. Mashuqur Rahman: *Recognition of English Characters by Syntactic Method of Pattern Recognition*: M. Sc. Engg. Thesis, 1985.Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

18. K. S. Fu.: Introduction, *Syntactic Pattern Recognition and Applications*: pp. 1 - 44, 163 - 181, Prince-Hall, New Jersey, 1982.

19. J. T. Tou and R. C. Gonzalez: *Pattern Recognition Principles*: 1981, Addision-Wesley Publishing Company, Reading, Massachusetts, U. S. A.

20. Rosenfeld, A.: *Digital Picture Analysis*: Edited by Rosenfeld A., Supriger - Verlag, Berlin, 1976.

21. Md. Mozammel Huq Azad Khan: *Optimal Realization of Bengali Key-board and Character Encoding for Computer Applications*: M. Sc. Engg. Thesis, Oct. 1986, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

22. K. S. Fu: *Recent Developments in Pattern Recognition*: IEEE Trans. on Computers, vol.: C - 29, October 1980.

23. Casey and Nagy: *Printed Chinese Character Recognition*: IEEE Trans. on Electronic Computers, vol.: EC-15, no. 1, February 1966, pp. 91 - 101.

24. William Stallings: *Approaches to Chinese Character Recognition*: Pattern Recognition, The Journal of the Pattern Recognition Society: vol 8, no. 2, pp 87 - 98.

25. J. L. Muerle and D. C. Allen: *Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene*: Pictorial Pattern Recognition: Thompson, Washington, D.C., 1968, pp 3 - 13.

26. T. Pavlidis: *Segmentation of Pictures and Maps Through Functional Approximation*: Comput. Graphics Image Process. 1, 360-72 (1972).

27. T. V. Robertson, K. S. Fu, and P. H. Swain: *Multispectral Image Partitioning*: Tech. Rept. TR-EE73-26. School of Electrical Engg., Purdue University, West Lafayette, Ind., 1973.

28. A. Klinger and C. R. Dyer: *Experiments on Picture Representation using Regular Decomposition*: Proc. 2nd Int. Joint Conf. Pattern Recognition, 1974, pp. 424-33.

29. S. L. Horowitz and T. Pavlidis: *Picture Segmentation by a Directed Split-and-merge Procedure*: Proc. 2nd Int. Joint Conf. Pattern Recognition, 1974, pp. 424 - 433.

APPENDIX A

APPENDIX A

REGRESSION ANALYSIS FOR FINDING THE CO-EFFICIENTS OF A SECOND ORDER EQUATION.

Comparing the magnitudes of the co-efficients of the equation $x = a + by + cy^2$, it is possible to make an idea about the curve that fits best to an array of N number of points. With $b = c = 0$, the best fit would be a straight line parallel to the y-axis; with b dominating, the best fit would be a sloping line, and with c dominating, the best fit would be parabolic in nature. An idea about other natures of the best fit can also be obtained by comparing the magnitude of a, b and c.

To get the co-efficients, regression analysis is performed. From

$$x = a + by + cy^2$$

the following equations can be obtained for N number of points

$$\Sigma x = aN + b\Sigma y + c\Sigma y^2$$

$$\Sigma xy = a\Sigma y + b\Sigma y^2 + c\Sigma y^3$$

$$\Sigma xy^2 = a\Sigma y^2 + b\Sigma y^3 + c\Sigma y^4$$

Co-efficients a, b and c can now be obtained by solving the above three simultaneous equations as:

$$a = \frac{\begin{vmatrix} \Sigma x & \Sigma y & \Sigma y^2 \\ \Sigma xy & \Sigma y^2 & \Sigma y^3 \\ \Sigma xy^2 & \Sigma y^3 & \Sigma y^4 \end{vmatrix}}{D}$$

$$b = \frac{\begin{vmatrix} N & \Sigma x & \Sigma y^2 \\ \Sigma y & \Sigma xy & \Sigma y^3 \\ \Sigma y^2 & \Sigma xy^2 & \Sigma y^4 \end{vmatrix}}{D}$$

$$c = \frac{\begin{vmatrix} N & \Sigma y & \Sigma y \\ \Sigma y & \Sigma y^2 & \Sigma xy \\ \Sigma y^2 & \Sigma y^3 & \Sigma xy^2 \end{vmatrix}}{D}$$

where,

$$D = \begin{vmatrix} N & \Sigma y & \Sigma y^2 \\ \Sigma y & \Sigma y^2 & \Sigma y^3 \\ \Sigma y^2 & \Sigma y^3 & \Sigma y^4 \end{vmatrix}$$

APPENDIX B

APPENDIX B

DICTIONARY SHOWING NUMERIC CODES CORRESPONDING TO ALL BANGLA ALPHABETIC CHARACTERS.

Character	Code	Character	Code	Character	Code
SharAow	453464	SharAaow	35553464	DirGhawE	56532643
SharAow	453564	SharAaow	35555464	DirGhawE	56532644
SharAow	454464	SharAaow	35563464	DirGhawE	56532645
SharAow	454564	SharAaow	35565464	DirGhawE	56552633
SharAow	455464	SharAaow	3453464	DirGhawE	56552634
SharAow	455564	RasshaE	534564	DirGhawE	56552635
SharAow	433464	RasshaE	534565	DirGhawE	56552643
SharAow	433564	RasshaE	564564	DirGhawE	56552644
SharAow	434464	RasshaE	564565	DirGhawE	56552645
SharAow	434564	RasshaE	5345644	RasshaOu	5643326
SharAow	435464	RasshaE	5345646	RasshaOu	5643426
SharAow	435564	RasshaE	5355644	RasshaOu	5653326
SharAow	553464	RasshaE	5355646	RasshaOu	5653426
SharAow	553564	RasshaE	5645644	DirghawOu	46433326
SharAow	554464	RasshaE	5645646	DirghawOu	46433426
SharAow	554564	RasshaE	5655644	DirghawOu	46434326
SharAow	555464	RasshaE	5655646	DirghawOu	46434426
SharAow	555564	DirGhawE	56422633	DirghawOu	46533326
SharAow	533464	DirGhawE	56422634	DirghawOu	46533426
SharAow	533564	DirGhawE	56422635	DirghawOu	46534326
SharAow	534464	DirGhawE	56422643	DirghawOu	46534426
SharAow	534564	DirGhawE	56422644	DirghawOu	56433326
SharAow	535464	DirGhawE	56422645	DirghawOu	56433426
SharAow	535564	DirGhawE	56432633	DirghawOu	56434326
SharAaow	34333464	DirGhawE	56432634	DirghawOu	56434426
SharAaow	34335464	DirGhawE	56432635	DirghawOu	56533326
SharAaow	34353464	DirGhawE	56432643	DirghawOu	56533426
SharAaow	34355464	DirGhawE	56432644	DirghawOu	56534326
SharAaow	34363464	DirGhawE	56432645	DirghawOu	56534426
SharAaow	34365464	DirGhawE	56452633	Rrie	33225344
SharAaow	34533464	DirGhawE	56452634	Rrie	33225346
SharAaow	34535464	DirGhawE	56452635	Rrie	33225364
SharAaow	34553464	DirGhawE	56452643	Rrie	33225366
SharAaow	34555464	DirGhawE	56452644	Rrie	33255344
SharAaow	34563464	DirGhawE	56452645	Rrie	33255346
SharAaow	34565464	DirGhawE	56522633	Rrie	33255364
SharAaow	35333464	DirGhawE	56522634	Rrie	33255366
SharAaow	35335464	DirGhawE	56522635	Rrie	33525344
SharAaow	35353464	DirGhawE	56522643	Rrie	33525346
SharAaow	35355464	DirGhawE	56522644	Rrie	33525364
SharAaow	35363464	DirGhawE	56522645	Rrie	33525366
SharAaow	35365464	DirGhawE	56532633	Rrie	33555344
SharAaow	35533464	DirGhawE	56532634	Rrie	33555346
SharAaow	35535464	DirGhawE	56532635	Rrie	33555364

Character	Code	Character	Code	Character	Code
Rrie	33555366	Kaw	453656	Ummaw	526526
Rrie	33625344	Kaw	456354	Ummaw	526566
Rrie	33625346	Kaw	456356	Ummaw	546326
Rrie	33625364	Kaw	456454	Ummaw	546366
Rrie	33625366	Kaw	456456	Ummaw	546426
Rrie	33655344	Kaw	456654	Ummaw	546466
Rrie	33655346	Kaw	456656	Ummaw	546526
Rrie	33655364	Kaw	553354	Ummaw	546566
Rrie	33655366	Kaw	553356	Chaw	456
A	5354	Kaw	553454	Chaw	436
A	5356	Kaw	553456	Chaw	556
A	5344	Kaw	553654	Chaw	536
A	5346	Kaw	553656	Chhaw	436644
Ooi	565354	Kaw	556354	Chhaw	436646
Ooi	565334	Kaw	556356	Jaw	555436
O	56456	Kaw	556454	Jaw	555446
O	53456	Kaw	556456	Jaw	555466
Oou	5656356	Kaw	556654	Jaw	555636
Oou	5656456	Kaw	556656	Jaw	555646
Oou	5656656	Khaw	325644	Jaw	555666
Oou	5654356	Khaw	325646	Jaw	555336
Oou	5654456	Gaw	3535	Jaw	555346
Oou	5654656	Ghaw	4356	Jaw	555366
Kaw	4653456	Ghaw	4354	Jaw	554436
Kaw	4653454	Ummaw	426326	Jaw	554446
Kaw	4653656	Ummaw	426366	Jaw	554466
Kaw	4653654	Ummaw	426426	Jaw	554636
Kaw	4653356	Ummaw	426466	Jaw	554646
Kaw	4653354	Ummaw	426526	Jaw	554666
Kaw	4656456	Ummaw	426566	Jaw	554336
Kaw	4656454	Ummaw	446326	Jaw	554346
Kaw	4656656	Ummaw	446366	Jaw	554366
Kaw	4656654	Ummaw	446426	Jaw	553436
Kaw	4656356	Ummaw	446466	Jaw	553446
Kaw	4656354	Ummaw	446526	Jaw	553466
Kaw	5653456	Ummaw	446566	Jaw	553636
Kaw	5653454	Ummaw	226326	Jaw	553646
Kaw	5653656	Ummaw	226366	Jaw	553666
Kaw	5653654	Ummaw	226426	Jaw	553336
Kaw	5653356	Ummaw	226466	Jaw	553346
Kaw	5653354	Ummaw	226526	Jaw	553366
Kaw	5656456	Ummaw	226566	Jaw	455436
Kaw	5656454	Ummaw	246326	Jaw	455446
Kaw	5656656	Ummaw	246366	Jaw	455466
Kaw	5656654	Ummaw	246426	Jaw	455636
Kaw	5656356	Ummaw	246466	Jaw	455646
Kaw	5656354	Ummaw	246526	Jaw	455666
Kaw	453354	Ummaw	246566	Jaw	455336
Kaw	453356	Ummaw	526326	Jaw	455346
Kaw	453454	Ummaw	526366	Jaw	455366
Kaw	453456	Ummaw	526426	Jaw	454436
Kaw	453654	Ummaw	526466	Jaw	454446

Character	Code	Character	Code	Character	Code
Jaw	454466	Jaw	4544346	Jaw	5545646
Jaw	454636	Jaw	4544336	Jaw	5545636
Jaw	454646	Jaw	4544366	Jaw	5545666
Jaw	454666	Jaw	4544326	Jaw	5545626
Jaw	454336	Jaw	4544246	Jaw	5545346
Jaw	454346	Jaw	4544236	Jaw	5545336
Jaw	454366	Jaw	4544266	Jaw	5545366
Jaw	453436	Jaw	4544226	Jaw	5545326
Jaw	453446	Jaw	4535646	Jaw	5545246
Jaw	453466	Jaw	4535636	Jaw	5545236
Jaw	453636	Jaw	4535666	Jaw	5545266
Jaw	453646	Jaw	4535626	Jaw	5545226
Jaw	453666	Jaw	4535346	Jaw	5544646
Jaw	453336	Jaw	4535336	Jaw	5544636
Jaw	453346	Jaw	4535366	Jaw	5544666
Jaw	453366	Jaw	4535326	Jaw	5544626
Jaw	4555646	Jaw	4535246	Jaw	5544346
Jaw	4555636	Jaw	4535236	Jaw	5544336
Jaw	4555666	Jaw	4535266	Jaw	5544366
Jaw	4555626	Jaw	4535226	Jaw	5544326
Jaw	4555346	Jaw	4534646	Jaw	5544246
Jaw	4555336	Jaw	4534636	Jaw	5544236
Jaw	4555366	Jaw	4534666	Jaw	5544266
Jaw	4555326	Jaw	4534626	Jaw	5544226
Jaw	4555246	Jaw	4534346	Jaw	5535646
Jaw	4555236	Jaw	4534336	Jaw	5535636
Jaw	4555266	Jaw	4534366	Jaw	5535666
Jaw	4555226	Jaw	4534326	Jaw	5535626
Jaw	4554646	Jaw	4534246	Jaw	5535346
Jaw	4554636	Jaw	4534236	Jaw	5535336
Jaw	4554666	Jaw	4534266	Jaw	5535366
Jaw	4554626	Jaw	4534226	Jaw	5535326
Jaw	4554346	Jaw	5555646	Jaw	5535246
Jaw	4554336	Jaw	5555636	Jaw	5535236
Jaw	4554366	Jaw	5555666	Jaw	5575266
Jaw	4554326	Jaw	5555626	Jaw	5535226
Jaw	4554246	Jaw	5555346	Jaw	5534646
Jaw	4554236	Jaw	5555336	Jaw	5534636
Jaw	4554266	Jaw	5555366	Jaw	5534666
Jaw	4554226	Jaw	5555326	Jaw	5534626
Jaw	4545646	Jaw	5555246	Jaw	5534346
Jaw	4545636	Jaw	5555236	Jaw	5534336
Jaw	4545666	Jaw	5555266	Jaw	5534366
Jaw	4545626	Jaw	5555226	Jaw	5534326
Jaw	4545346	Jaw	5554646	Jaw	5534246
Jaw	4545336	Jaw	5554636	Jaw	5534236
Jaw	4545366	Jaw	5554666	Jaw	5534266
Jaw	4545326	Jaw	5554626	Jaw	5534226
Jaw	4545246	Jaw	5554346	Jhaw	355344
Jaw	4545236	Jaw	5554336	Jhaw	355346
Jaw	4545266	Jaw	5554366	Jhaw	355364
Jaw	4545226	Jaw	5554326	Jhaw	355366
Jaw	4544646	Jaw	5554246	Jhaw	345344
Jaw	4544636	Jaw	5554236	Jhaw	345346
Jaw	4544666	Jaw	5554266	Jhaw	345364
Jaw	4544626	Jaw	5554226	Jhaw	345366

Character	Code	Character	Code	Character	Code
Niow	523655344	Niow	253645344	Dhaow	665534
Niow	523655346	Niow	253645346	Dhaow	665536
Niow	523655354	Niow	253645354	Dhaow	664534
Niow	523655356	Niow	253645356	Dhaow	664536
Niow	523655644	Niow	253645644	Dhaow	663534
Niow	523655646	Niow	253645646	Dhaow	663536
Niow	523655654	Niow	253645654	Dhaow	655534
Niow	523655656	Niow	253645656	Dhaow	655536
Niow	523645344	Taw	56433	Dhaow	654534
Niow	523645346	Taw	56434	Dhaow	654536
Niow	523645354	Taw	56533	Dhaow	653534
Niow	523645356	Taw	56534	Dhaow	653536
Niow	523645644	Taw	564356	Dhaow	36434
Niow	523645646	Taw	564352	Dhaow	36436
Niow	523645654	Taw	565356	dnaw	4354
Niow	523645656	Taw	565352	dnaw	5354
Niow	553655344	Thaw	5426	paw	35623
Niow	553655346	Thaw	5456	paw	35622
Niow	553655354	Thaw	5436	paw	35633
Niow	553655356	Thaw	3426	paw	35632
Niow	553655644	Thaw	3456	paw	35223
Niow	553655646	Thaw	3436	paw	35222
Niow	553655654	Ddaw	53426	paw	35233
Niow	553655656	Ddaw	43426	paw	35232
Niow	553645344	Ddhaw	4352	phaw	435656
Niow	553645346	Ddhaw	4356	phaw	435654
Niow	553645354	Ddhaw	4552	phaw	435356
Niow	553645356	Ddhaw	4556	phaw	435354
Niow	553645644	Ddhaw	5352	baw	46534
Niow	553645646	Ddhaw	5356	baw	46536
Niow	553645654	Ddhaw	5552	baw	4536
Niow	553645656	Ddhaw	5556	baw	4534
Niow	223655344	Ddhaw	453	bhaw	44526
Niow	223655346	Ddhaw	433	bhaw	54526
Niow	223655354	Ddhaw	553	Maow	43654
Niow	223655356	Ddhaw	533	Maow	53654
Niow	223655644	ModiNanNaw	3532	Ontasta.jaw	4356
Niow	223655646	ModiNanNaw	3535	Ontasta.jaw	4354
Niow	223655654	Taow	4456	Raw	465345
Niow	223655656	Taow	5456	Raw	465365
Niow	223645344	Thaow	35646	Raw	565345
Niow	223645346	Thaow	35644	Raw	565365
Niow	223645354	Daow	4323	Raw	55345
Niow	223645356	Daow	323	Raw	55365
Niow	223645644	Dhaow	565534	Raw	45345
Niow	223645646	Dhaow	565536	Raw	45365
Niow	223645654	Dhaow	564534	Law	4352435
Niow	223645656	Dhaow	564536	Law	4352445
Niow	253655344	Dhaow	563534	Law	4352465
Niow	253655346	Dhaow	563536	Law	4352335
Niow	253655354	Dhaow	555534	Law	4352345
Niow	253655356	Dhaow	555536	Law	4352365
Niow	253655644	Dhaow	554534	Law	4352635
Niow	253655646	Dhaow	554536	Law	4352645
Niow	253655654	Dhaow	553534	Law	4352665
Niow	253655656	Dhaow	553536	Law	4355435

Character	Code
Law	4355445
Law	4355465
Law	4355335
Law	4355345
Law	4355365
Law	4355635
Law	4355645
Law	4355665
Law	4322435
Law	4322445
Law	4322465
Law	4322335
Law	4322345
Law	4322365
Law	4322635
Law	4322645
Law	4322665
Law	4325435
Law	4325445
Law	4325465
Law	4325335
Law	4325345
Law	4325365
Law	4325635
Law	4325645
Law	4325665
Law	5352435
Law	5352445
Law	5352465
Law	5352335
Law	5352345
Law	5352365
Law	5352635
Law	5352645
Law	5352665
Law	5355435
Law	5355445
Law	5355465
Law	5355335
Law	5355345
Law	5355365
Law	5355635
Law	5355645
Law	5355665
Law	5322435
Law	5322445
Law	5322465
Law	5322335
Law	5322345
Law	5322365
Law	5322635
Law	5322645
Law	5322665
Law	5325435
Law	5325445
Law	5325465

Character	Code
Law	5325335
Law	5325345
Law	5325365
Law	5325635
Law	5325645
Law	5325665
TalibbaSha	342234
TalibbaSha	342235
TalibbaSha	342534
TalibbaSha	342535
TalibbaSha	342334
TalibbaSha	342335
TalibbaSha	345234
TalibbaSha	345235
TalibbaSha	345534
TalibbaSha	345535
TalibbaSha	345334
TalibbaSha	345335
TalibbaSha	344234
TalibbaSha	344235
TalibbaSha	344534
TalibbaSha	344535
TalibbaSha	344334
TalibbaSha	344335
TalibbaSha	352234
TalibbaSha	352235
TalibbaSha	352534
TalibbaSha	352535
TalibbaSha	352334
TalibbaSha	352335
TalibbaSha	355234
TalibbaSha	355235
TalibbaSha	355534
TalibbaSha	355535
TalibbaSha	355334
TalibbaSha	355335
TalibbaSha	354234
TalibbaSha	354235
TalibbaSha	354534
TalibbaSha	354535
TalibbaSha	354334
TalibbaSha	354335
TalibbaSha	322234
TalibbaSha	322235
TalibbaSha	322534
TalibbaSha	322535
TalibbaSha	322334
TalibbaSha	322335
TalibbaSha	325234
TalibbaSha	325235
TalibbaSha	325534
TalibbaSha	325535
TalibbaSha	325334
TalibbaSha	325335
TalibbaSha	324234
TalibbaSha	324235

Character	Code
TalibbaSha	324534
TalibbaSha	324535
TalibbaSha	324334
TalibbaSha	324335
ModinasSha	43536
ModinasSha	43534
DontasSha	43634
DontasSha	43635
DontasSha	43633
DontasSha	43654
DontasSha	43655
DontasSha	43653
Haw	45644
Haw	45646
Haw	4565
Haw	465644
Haw	465634
Haw	465664
Haw	465444
Haw	465434
Haw	465464
Haw	466644
Haw	466634
Haw	466664
Haw	466444
Haw	466434
Haw	466464
Haw	455644
Haw	455634
Haw	455664
Haw	455444
Haw	455434
Haw	455464
Haw	456644
Haw	456634
Haw	456664
Haw	456444
Haw	456434
Haw	456464
DhoRaw	434265
DhoRaw	433265
DosinnaRaw	43565
DosinnaRaw	43525
DosinnaRaw	4335
DosinnaRaw	5335
ontasta	53545
ontasta	53565
ontasta	43545
ontasta	43565
KhondaTaw	5656
KhondaTaw	5654
KhondaTaw	5646
KhondaTaw	5644
KhondaTaw	5636
KhondaTaw	5634
KhondaTaw	563

Character	Code
KhondaTaw	564
KhondaTaw	565
Unoshar	565
Bisogha	5656
Bisogha	5652
Bisogha	5256
Bisogha	5252
ChandraBindo	442
ChandraBindo	443
ChandraBindo	444
ChandraBindo	445
ChandraBindo	422
ChandraBindo	423
ChandraBindo	424
ChandraBindo	425
ChandraBindo	452
ChandraBindo	453
ChandraBindo	454
ChandraBindo	455
ChandraBindo	432
ChandraBindo	433
ChandraBindo	434
ChandraBindo	435
ChandraBindo	462
ChandraBindo	463
ChandraBindo	464
ChandraBindo	465
ChandraBindo	542
ChandraBindo	543
ChandraBindo	544
ChandraBindo	545
ChandraBindo	522
ChandraBindo	523
ChandraBindo	524
ChandraBindo	525
ChandraBindo	552
ChandraBindo	553
ChandraBindo	554
ChandraBindo	555
ChandraBindo	532
ChandraBindo	533
ChandraBindo	534
ChandraBindo	535
ChandraBindo	562
ChandraBindo	563
ChandraBindo	564
ChandraBindo	565

APPENDIX C

APPENDIX C

MAIN PROGRAM FOR BANGLA CHARACTER RECOGNITION

```
# include <stdio.h>
# include <stdlib.h>
# include <graphics.h>
# include <float.h>
# include <math.h>

char LetterCode[10];
float StrokePercent[10];
int LetterHeight, LetterWidth, CodeLength;
char StrokeCode;                    /* Code for a particular   */
                                    /* stroke. */
float MorphPercent;

FILE *OutFile;
char FileName[31];                /* File name from the user  */

int  KKStrokeNumber ;
unsigned char *Letter, *StartOfLetter;  /* For storing the   */
                                        /* actual letter bit */
                                        /* per pixel map     */
int *Strokes, *StartOfStrokes, *CurrentStroke;
                                          /* Various pointers      */
                                          /* to the stroke         */
                                          /* coordinates stored    */
                                          /* in memory.            */
unsigned char StrokeNumber;               /* For counting the      */
                                          /* strokes               */

void GetLetter(void);
void GetLetterContour(void);
void FilterOut(void);
void StoreStrokes(void);
void GetLetterCode(void);
void FindStrokeCode(void);
void SetGraphicsMode(void);
void DisplayLetter(unsigned XTop, unsigned YTop);
void KeepInFile(void);
void CoordinatesToFile(void);
void setprn(void);
void resetprn(void);
void initline( int gcol);
void toPrint(int x1,int y1,int x2,int y2);
void RecognizeChar(void);
```

```
void main(void) {

  /* Read the raw structure of the letter from file LETTER.TMP */

  GetLetter();
/* DisplayLetter(50,50);  */
/* KeepInFile(); */

/* Read the structure of the actual letter from memory location */
/* 'Letter' and then mark the white to black transition strokes */

  GetLetterContour();
/* DisplayLetter(100,50);  */
/* KeepInFile(); */

/* Filter out stray points */
  FilterOut();
/* DisplayLetter(100,50);*/
/* KeepInFile(); */

/* Save the stroke coordinates to memory starting from 'Stroke' */
  StoreStrokes();

/* From the stroke coordinates, find the letter code */
  GetLetterCode();

  printf ("\n\n\t Letter code is %s", LetterCode);
        printf("\n\n\n\t");
  RecognizeChar();
}

void GetLetter(void) {

/* This function reads the structure of letter from the          */
/* file LETTER.TMP to the array RawLetter [][]. This file        */
/* was created by scanning the letter by a handy scanner.        */
/* It contains the full screen bit map in 348 * 720/8 bytes      */
/* From the full screen bit map this function extracts the       */
/* the area containing actual letter. and stores it in           */
/* the memory location starting from 'Letter'.                   */

/* External variables used are:                  */
/*   int LetterHeight, LetterWidth               */
/*   unsigned char *Letter, *StartOfLetter.      */

  #define SCREENHEIGHT 348              /* 348 lines.            */
  #define SCREENWIDTH 720/8             /* 720 pixels per line   */
                                        /* Stored in 720/8 bytes.*/
  #define MAXBLANKLINE 25

  FILE *InFile;                         /* Input file pointer    */

  int Left, Right, Top, Bottom;         /* Letter boundary       */
                                        /* markers.              */
  int x, y;                             /* Coordinate variables. */
  int k;                                /* Scratch variable.     */
  unsigned char BlackFound;             /* Line containing       */
                                        /* black points found flag.*/
```

```
unsigned char BlankLineNo=0;

unsigned char RawLetter [SCREENHEIGHT] [SCREENWIDTH];

/* Total 256 elements in the array below which indicates  */
/* the number of ones in the corresponding binary form of */
/* the index. */
unsigned char NoOfBlacks [] = {
0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4,1,2,2,3,2,3,3,4,2,3,3,4,3,4,
4,5,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,4,5,
4,5,5,6,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,
4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,4,5,4,5,5,6,
4,5,5,6,5,6,6,7,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,
4,5,3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,4,5,
4,5,5,6,4,5,5,6,5,6,6,7,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,
4,5,4,5,5,6,4,5,5,6,5,6,6,7,3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
4,5,5,6,5,6,6,7,5,6,6,7,6,7,7,8  };

/* Assign all letter boundaries to zero */
Left = Right = Top = Bottom = 0;

/* Open file LETTER.TMP. Issue message if could not be opened */
if ((InFile = fopen("LETTER.TMP","rb")) == NULL)
  {
      perror("\n\a\aSorry, file LETTER.TMP could not be opened");
      exit(1);
  }

/* With the input file open, transfer each line to array, */
/* RawLetter. Negate bitwise before transfer.             */
BlankLineNo=0;
for (y = 0; y < SCREENHEIGHT; y++) {
      BlackFound = 0;
      for (x = 0; x < SCREENWIDTH; x++) {
              RawLetter [y] [x] = ~(getc(InFile));

        /* See if the character can be treated as  */
        /* significantly black. A byte is taken as */
        /* black if it has more than one black (1) */
        /* points */
              if (NoOfBlacks[RawLetter [y] [x]] > 1) {
                      BlackFound = 1;
                      if (!Left)
                         Left = x;
                      else
                         if (x < Left) Left = x;

                      if (x > Right) Right = x;
              }
        }

      /* If a line containing black points has been found    */
      /* and Top has not been assigned (equal to zero) it is */
      /* the top margin line */

      if ((BlackFound == 1) && (Top == 0))
              Top = y;
      if((BlackFound==1)&&(BlankLineNo > 0))
```

```
                BlankLineNo=0;

        /* But if a black line has been found and Top has been */
        /* assigned (non zero) the previous line was the bottom */
        /* margin line */

        if ((BlackFound == 0) && (Top != 0)) {
           if(BlankLineNo < MAXBLANKLINE)
              BlankLineNo++;
           else {
              Bottom = y - BlankLineNo;
              break;
           }
        }
    }

    /* Calculate letter dimensions */
    LetterHeight = (Bottom - Top) + 1;
    LetterWidth  = ((Right - Left) + 1);

    /* Allocate memory for the actual letter, which will contain */
    /* the stroke bit map of the marked boundary */

    Letter=(char *)malloc(LetterHeight * LetterWidth * sizeof(char));
    StartOfLetter = Letter;                /* Save starting position */

    if (Letter == NULL) {
       printf("\a\a\nMemory allocation failure.");
       printf("\nLetter is too large in size for recognition");
       printf("\nor stray points on paper, or there are many ");
       printf("\nmemory resident programs");
       exit(1);
    }

    for (y = Top; y <= Bottom; y ++) {
        for (x = Left; x <= Right; x ++) {
           *Letter = RawLetter[y][x];
           Letter ++;
        }
    }
}

void GetLetterContour(void) {

    /* This function reads the structure of letter from the       */
    /* memory location 'Letter' which contains the actual bit map */
    /* of the input character. This memory was created by the     */
    /* function GetLetter(). This function gets the white         */
    /* to black transition stroke of the letter by scanning each  */
    /* line of this area from left to right and stores them in    */
    /* the memory location starting from 'Letter'.                */

    /* External variables used are: */
    /*   int LetterHeight, LetterWidth    */
    /*   unsigned char *Letter, *StartOfLetter;  */

         int  x, y;                        /* Coordinate variables */
         char temp;
```

```
        unsigned char TransitionState;  /* Byte for indicating  */
                                        /* whether we are       */
                                        /* already in the middle*/
                                        /* of a black           */
                                        /* region, meaning      */
                                        /* transition started   */
                                        /* or a black           */
                                        /* point found indicates*/
                                        /* white to black       */
                                        /* transition, meaning  */
                                        /* not started.         */
                                        /* f0h = 1111 0000,     */
                                        /* transition started,  */
                                        /* 00h = 0000 0000,     */
                                        /* not started          */

/* Elements in the array below indicates the corresponding     */
/* elements to the characters in a RawLetter line when         */
/* scanned from left. In this table it has been assumed        */
/* that transition has 'not started' as explained earlier      */
/* 'Started' cases are also handled by this table with the     */
/* aid of the 'TransitionState' flag */

unsigned char TransitionTable [] = {
        0x00,0x01,0x02,0x02,0x04,0x05,0x04,0x04,0x08,0x09,
        0x0A,0x0A,0x08,0x09,0x08,0x08,0x10,0x11,0x12,0x12,
        0x14,0x15,0x14,0x14,0x10,0x11,0x12,0x12,0x10,0x11,
        0x10,0x10,0x20,0x21,0x22,0x22,0x24,0x25,0x24,0x24,
        0x28,0x29,0x2A,0x2A,0x28,0x29,0x28,0x28,0x20,0x21,
        0x22,0x22,0x24,0x25,0x24,0x24,0x20,0x21,0x22,0x22,
        0x20,0x21,0x20,0x20,0x40,0x41,0x42,0x42,0x44,0x45,
        0x44,0x44,0x48,0x49,0x4A,0x4A,0x48,0x49,0x48,0x48,
        0x50,0x51,0x52,0x52,0x54,0x55,0x54,0x54,0x58,0x59,
        0x5A,0x52,0x50,0x51,0x50,0x50,0x40,0x41,0x42,0x42,
        0x44,0x44,0x44,0x44,0x48,0x49,0x4A,0x4A,0x48,0x49,
        0x48,0x48,0x40,0x41,0x42,0x42,0x44,0x45,0x44,0x44,
        0x40,0x41,0x42,0x42,0x40,0x41,0x40,0x40,0x80,0x81,
        0x82,0x82,0x84,0x85,0x84,0x84,0x88,0x89,0x8A,0x8A,
        0x88,0x89,0x88,0x88,0x90,0x91,0x92,0x92,0x94,0x95,
        0x94,0x94,0x90,0x91,0x92,0x92,0x90,0x91,0x90,0x90,
        0xA0,0xA1,0xA2,0xA2,0xA4,0xA5,0xA4,0xA4,0xA8,0xA9,
        0xAA,0xAA,0xA8,0xA9,0xA8,0xA8,0xA0,0xA1,0xA2,0xA2,
        0xA4,0xA5,0xA4,0xA4,0xA0,0xA1,0xA2,0xA2,0xA0,0xA1,
        0xA0,0xA0,0x80,0x81,0x82,0x82,0x84,0x85,0x84,0x84,
        0x84,0x89,0x8A,0x8A,0x88,0x89,0x88,0x88,0x90,0x91,
        0x92,0x92,0x94,0x95,0x94,0x94,0x98,0x99,0x92,0x9A,
        0x90,0x91,0x90,0x90,0x80,0x81,0x82,0x82,0x84,0x85,
        0x84,0x84,0x88,0x89,0x8A,0x8A,0x88,0x89,0x88,0x88,
        0x80,0x81,0x82,0x82,0x84,0x85,0x84,0x84,0x80,0x81,
        0x82,0x82,0x80,0x81,0x80,0x80 };

        Letter = StartOfLetter;         /* Point to starting    */

for (y = 0; y < LetterHeight; y ++) {
        TransitionState = 0;            /* Set initial state to */
                                        /* not started state.   */
```

```
        for (x = 0; x < LetterWidth; x ++)
        {
           /* Get the 0 to 1 transition strokes using table */
           /* and flag and save in Letter */

           temp = *Letter;
           *Letter = (TransitionTable [temp]) & ~(TransitionState);
           TransitionState = (temp) << 7;
           Letter ++;
        }
    }
}

void FilterOut(void) {

   /* This function filters out all the stray points present in     */
   /* the contour map of the character to be recognized. Sometimes, */
   /* there may be loss of significant bits present in byte         */
   /* boundary in a single row. But this phenomenon, of very        */
   /* low probability, is allowed to speed up the process.          */
   /* This loss, if any, is later on remedied out when we merge     */
   /* strokes later on, which have their start points in            */
   /* the near vicinity of the end points of other strokes.         */

   /* External variables used are:                 */
   /*    int LetterHeight, LetterWidth             */
   /* unsigned char *Letter, *StartOfLetter        */

   int x, y;                              /* Coordinate variables */

   unsigned char Temp;                    /* Scratch variables    */

   /* Elements in the array below are the filter masks for       */
   /* filtering out all stray points present in the bit map      */
   /* of the strokes of the character to be recognized. Each     */
   /* element consists of the binary form of the element         */
   /* offset with three 1's at the left and right side of the    */
   /* 1's. For example element 20 (= 0001 0100B) is 1111 1111B)  */
   /* = 0xFF. Spilled out 1's at the left and right boundaries   */
   /* are simply discarded, giving array element for 33          */
   /* (= 0010 0001B) to be 1111 1111B = 0xFF. For elements       */
   /* with two or more consecutive 1's in the binary form of     */
   /* offset, the array element is simply dummy 0. The array     */
   /* has got a total of 171 elements  for 0 to 170,             */
   /* since numbers from 171 to 255 all have two or more         */
   /* consecutive 1's in their binary form.                      */

   unsigned char FilterMask [] = {
   0x00,0x0F,0x1F,0x00,0x3F,0x3F,0x00,0x00,0x7F,0x7F,0x7F,0x00,
   0x00,0x00,0x00,0x00,0xFE,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFC,0xFF,0xFF,0x00,
   0xFF,0xFF,0x00,0x00,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0xF8,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,
   0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0xF8,0xFF,0xFF,0x00,
   0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF0,0xFF,0xFF,0x00,
0xFF,0xFF,0x00,0x00,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
0xFE,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xFC,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0x00,
0xFF,0xFF,0xFF };

/* Filter out stand alone 1's, that is 1's which have no      */
/* 1 any one of its fourteen neighbor points, namely, top     */
/* left left left, top left left, top left, top, top right,   */
/* top right right, top right right right, bottom left left   */
/* left, bottom left left, bottom left, bottom, bottom right  */
/* and bottom right right, bottom right right right.          */
/* Filtering out is performed by anding each byte with the    */
/* mask of its top and bottom byte and oring these two        */
/* results. The result is stored in the byte location.        */
/* For the first and last line only the bottom and top bytes  */
/* respectively are taken.  */

Letter = StartOfLetter;                /* Point to starting */
                                       /* location */

/* For the first line, i.e. y = 0, take the bottom */
/* bytes only */

for (x = 0; x < LetterWidth; x ++) {
   *Letter = (FilterMask [(*(Letter+LetterWidth))])&(*Letter);
   Letter ++;
}

/* For the second line to the second last line i.e. y = 1 */
/* to y = LetterHeight - 2, (lines are counted from zero) */
/* take the top and bottom bytes. */

for (y = 1; y <= LetterHeight - 2; y ++) {
   for (x = 0; x < LetterWidth; x ++) {
       Temp = *Letter;
       *Letter = ((FilterMask[*(Letter - LetterWidth)])& Temp)|
                 ((FilterMask[*(Letter + LetterWidth)])& Temp);
       Letter ++;
   }
}

/* For the last line, i.e. y = LetterHeight - 1 (lines are */
/* counted from zero), take the top bytes only */

for (x = 0; x < LetterWidth; x ++) {
   *Letter = (FilterMask [(*(Letter - LetterWidth))])&(*Letter);
   Letter ++;
}

}
```

```
void StoreStrokes(void) {

  /* This function stores the stroke coordinates in the          */
  /* memory location Strokes. This memory location is            */
  /* composed of LetterHeight plus two multiplied by             */
  /* MAXSTROKES bytes. The first two bytes of each unit          */
  /* of LetterHeight plus two bytes is for storing the           */
  /* start and end y coordinates of the stroke. The rest         */
  /* LetterHeight bytes are for storing the x coordinates.       */
  /* Since strokes are seldom LetterHeight bytes long,           */
  /* there will be unused bytes in each unit, which are          */
  /* left undefined. */
  /* Storing of stroke coordinates is done by scanning the       */
  /* Letter lines from left to right starting from the           */
  /* upper left corner. Whenever a black point is found          */
  /* scanning proceeds to the bottom points for finding          */
  /* continued points. Points are assumed continuous when        */
  /* a point is within three bits of its immediate               */
  /* upper line black. If a discontinuity occurs, scanning       */
  /* proceeds to the next line with the assumption that          */
  /* the void in the previous line was due to the presence       */
  /* of 'horizontal bends' in the original letter. If the        */
  /* next line gives continuity, a black point is assumed        */
  /* in the previous line and its coordinates stored.            */
  /* Otherwise, the stroke is assumed to have terminated         */
  /* and its end y coordinate is stored. All black points        */
  /* of the stroke are nulled out to 0, during time of           */
  /* coordinate storing so that these are not considered         */
  /* again when scanning again starting from the upper left      */
  /* corner. */

  # define  MAXSTROKES 20
  unsigned char Position;                 /* Stores position of */
                                          /* black point found. */
  unsigned char NextPosition;             /* Stores position of */
                                          /* the black point found */
                                          /* in the next line */
        char VoidVoid;

  unsigned char Mask, Mask1, Mask2, Mask3;  /* Continuity checker  */

  unsigned char CurrentByte, NextByte;      /* For saving the byte */
                                            /* under consideration */
                                            /* and next line byte. */

  /* Total 256 elements in the array below indicates the number */
  /* of ones in the corresponding binary form of the index.     */

  unsigned char NoOfBlacks [] = {
  0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4,1,2,2,3,2,3,3,4,2,3,3,4,3,4,
  4,5,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,4,5,
  4,5,5,6,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,4,5,3,4,
  4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,4,5,4,5,5,6,
  4,5,5,6,5,6,6,7,1,2,2,3,2,3,3,4,2,3,3,4,3,4,4,5,2,3,3,4,3,4,
  4,5,3,4,4,5,4,5,5,6,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,4,5,
  4,5,5,6,4,5,5,6,5,6,6,7,2,3,3,4,3,4,4,5,3,4,4,5,4,5,5,6,3,4,
  4,5,4,5,5,6,4,5,5,6,5,6,6,7,3,4,4,5,4,5,5,6,4,5,5,6,5,6,6,7,
  4,5,5,6,5,6,6,7,5,6,6,7,6,7,7,8 };
```

```
/* Elements in the array below indicates the positions of      */
/* left most 1's in the binary representations of element      */
/* offsets. For example, for 4 = 0000 0100B the array element */
/* is 5 indicating a 1 in the 5th location from left           */
/* (start at 0). Again, for 21 = 0001 0101B, the element       */
/* is 3. For numbers with two or more consecutive 1's in       */
/* binary form (like 31 = 0001 1111B) elements are 0's.        */
/* The array  contains 171 elements for 0 to 170.              */
/* Numbers from 171 to 255 all have two or more                */
/* consecutive 1's in their binary form. Thus the array        */
/* has got one hundred and seventy one elements.               */

unsigned char BlackPosition [] = {
      0,7,6,0,5,5,0,0,4,4,4,0,0,0,0,0,3,3,0,0,
      3,3,0,0,0,0,0,0,0,0,0,0,2,2,2,0,2,2,0,0,
      2,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,1,1,1,0,1,1,0,0,1,1,1,0,0,0,0,0,
      1,1,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0 };

/* Elements in the table below are the masks required for      */
/* checking stroke continuity. The table serves a three fold   */
/* purpose. When checking continuity in the just below bytes,  */
/* it has to be used in three element group. Elements in each  */
/* group indicates the various masks required for diffrent     */
/* scanning ranges. To get mask for such cases group is to be  */
/* selected by the position of leftmost 1 in the binary        */
/* representation of the byte in the upper line. And then      */
/* element in the group will be selected by the left or right  */
/* check range. The first element of each group is the mask    */
/* which checks the continuity of 1 in the left or right bit   */
/* position of the next line. The second element of each group */
/* is the mask which checks the continuity of 1 in the second  */
/* left or second right bit position of the next line. The third */
/* element of each group is the mask which checks the continuity */
/* of 1 in the third left or third right bit position of the next*/
/* line. For example, element 2 of group 5 (element seventeen) */
/* is 0x3F (0011 1111B). This is the mask for byte 0000 0100 with*/
/* an assigned left and  right check range of 3.               */
/* There are a total of sixty four bytes in this array.        */

unsigned char MaskTable [] = {
  0xC0,0xE0,0xF0,0xE0,0xF0,0xF8,0x70,0xF8,0xFC,0x38,0x7C,0xFE,
  0x1C,0x3E,0x7F,0x0E,0x1F,0x3F,0x07,0x0F,0x1F,0x03,0x07,0x0F
};

int x, y;                    /* Coordinate variables */
int MaxStrokeLength;         /* Maximum length of a  */
                             /* stroke */

StrokeNumber = 0;

/* Maximum length of memory location for storing stroke */
/* coordinates is LetterHeight plus two. Two extra bytes */
```

```
  /* are for keeping the start and end y-coordinates */

  MaxStrokeLength = LetterHeight + 2;

  /* Allocate memory for storing the stroke coordinates */
  Strokes = (int *) malloc(MaxStrokeLength * MAXSTROKES
                                  * sizeof(int));

  StartOfStrokes = Strokes;              /* Save starting position */

  if (Strokes == NULL) {
      printf("\a\a\nMemory allocation failure.");
      printf("\nLetter is too large in size for recognition");
      printf("\nor stray points on paper, or there are many ");
      printf("\nmemory resident programs");
      exit(1);
  }

        /* Get name of the output file from user */
/*      clrscr();
        printf("Enter file name in which to store the letter: ");
        scanf("%s", FileName);

        if ((OutFile = fopen(FileName, "w")) == NULL)
                {
                perror("\n\a\aSorry, file could not be created");
                exit(1);
                }
        fprintf(OutFile, ".lh6\n.op \n.mt1 \n.mb1 \n.lm1");
*/

  /* Start a loop for scanning all the bytes again and again */
  for (;;)
  {
    Letter = StartOfLetter;

    /* Scan all the bytes starting from top left for */
    /* finding a 'black' byte */
    for (y = 0; y < LetterHeight; y ++)
    {
       for (x = 0; x < LetterWidth; x ++)
       {
       /* Check if the byte contains black dots */
          if (*Letter != 0)
          /* If black found, */
          {
             /* record its position if there is one '1' in the
               current byte  */

             Position = BlackPosition [*Letter];

             /* isolate the leftmost 1, */

             CurrentByte = 1 << (7 - Position);

             /* zero out this 1 in Letter, */
             *Letter = (*Letter) & ~(CurrentByte);
             /* and go to process it */
```

```
            goto StartFound;
        }
        Letter ++;
      }
    }
    /* When execution have reached this statement it means    */
    /* that blacks present in the letter have been processed */
    /* for strokes, so exit the for loop */

    break;


    /* Check if the maximum allowable number of strokes */
    /* has exceeded */

  StartFound:

     if (StrokeNumber > MAXSTROKES)
     /* Inform user if exceeded */
     {
         clrscr();
         printf ("\a\a\a\a\Letter has broken into very many ");
         printf ("elements due to poor construction ");
         printf ("\or improper scanning.\n\nCan't");
         printf (" process further.\nSorry.");

        /*              fclose(OutFile); */

         exit (1);
     }


     /* Process the black point found, which is the start */
     /* point of a stroke */
     /* Set the various memory pointers for the new stroke */

     Strokes = CurrentStroke = StartOfStrokes +
                               StrokeNumber * MaxStrokeLength;
     /* Save starting y location of stroke */

     *Strokes = (LetterHeight-1)-y;

/*      fprintf(OutFile, "\n ********************");
        fprintf(OutFile, "\n Stroke number is %d", StrokeNumber);
*/

     /* Save starting x coordinate */
     Strokes ++;
     Strokes ++;
     *Strokes = x * 8 + Position;

/*      fprintf(OutFile, "\n  X: %3d, Y: %3d", x * 8 + Position,
                                      (LetterHeight-1)-y);
*/
     /* Now start the loop for finding out the rest of the */
     /* points of the stroke just started. Loop exits      */
     /* when, continuity ceases or bottom line of          */
     /* letter reached */
```

```
for (; y < LetterHeight;) {

    /* Make Letter point to the byte just below    */
    /* the byte whose continuity is being checked. */

    Letter += LetterWidth;
    /* Point to next line */
    y ++;
    /* Check for bit just below the 'black' bit */
    /* under consideration */

    NextByte = *Letter;
    if (y == LetterHeight)
    {
       y --;
       goto EndStroke;
    }
    if (CurrentByte & NextByte)
    {
       if (NoOfBlacks[NextByte] == 2)
           NextPosition = Position;
           goto SaveX;
    }

    Mask1 = MaskTable[Position * 3     ];
    Mask2 = MaskTable[Position * 3 + 1 ];
    if(LetterHeight > 60)
        Mask3 = MaskTable[Position * 3 + 2 ];
    else
        Mask3 = 0;

    if((Mask1 & NextByte)||(Mask2 & NextByte)||\
                           (Mask3 & NextByte)) {
       if (NoOfBlacks[NextByte] == 2) {
           if(Position > 3)
               NextPosition = BlackPosition[NextByte & 0x0F];
           else
               NextPosition = BlackPosition[NextByte];
       }
       goto SaveX;
    }

    /* If the above check failed, check */
    /* continuity in left and right bytes */

    if(x!=0 && ((Position==0)||(Position==1)||((LetterHeight>60)\
                                  &&(Position==2))))
    {
        NextByte= *(Letter-1);
        if((NextByte & 0x01)||\
          ((Position==1)&&(LetterHeight>60)&&(NextByte & 0x03))||
          ((Position==0)&&((NextByte & 0x03)||((LetterHeight>60)\
          &&(NextByte & 0x07)))))
          {
             x--;
             Letter--;
             if (NoOfBlacks[NextByte] == 2)
                 NextPosition = BlackPosition[NextByte & 0x07];
```

```
                goto SaveX;
        }
}
if(x!=(LetterWidth-1) && (((LetterHeight>60)&&(Position==5))\
        ||(Position==6)||(Position==7)))
{
   NextByte= *(Letter+1);
   if((NextByte & 0x80)||\
     ((Position==6)&&(LetterHeight>60)&&(NextByte & 0xC0))||
     ((Position==7)&&((NextByte & 0xC0)||((LetterHeight>60)\
     &&(NextByte & 0xE0)))))
   {
      x++;
      Letter++;
      if (NoOfBlacks[NextByte] == 2)
         NextPosition = BlackPosition[NextByte];
      goto SaveX;
   }
}
/* When program has reached this point it      */
/* indicates that no continuity has been       */
/* found in the previous checks. So    */
/* assume that the line was void due           */
/* to bit loss in stroke scanning and try the  */
/* next line. But before that check if bottom  */
/* edge of letter is reached, save             */
/* coordinates and exit if yes.                */

/* Check this continuity only if LetterHeight is
greater than 40. */

if((y<LetterHeight-1)&&(LetterHeight>30)&&\
        ((LetterWidth*8)>30))
{

/* Check for bit just below the 'black' bit */
/* under consideration */

/* Make Letter point to the byte two lines below */
/* the byte whose continuity is being checked */

Letter += LetterWidth;

/* Point to next line */

y ++;

/* Check for bit below the 'black' bit */
/* under consideration */

NextByte = *Letter;

if (CurrentByte & NextByte) {
   if (NoOfBlacks[NextByte] == 2)
       NextPosition = Position;
   goto FillVoid;
}
```

```
    if((Mask1 & NextByte)||(Mask2 & NextByte)) {
       if (NoOfBlacks[NextByte] == 2){
          if(Position > 3)
             NextPosition = BlackPosition[NextByte & 0x0F];
          else
             NextPosition = BlackPosition[NextByte];
       }
              goto FillVoid;
    }

    /* If the above check failed, check     */
    /* continuity in left and right bytes. */

    if(x!=0 && ((Position==0)||(Position==1)))
    {
       NextByte= *(Letter-1);
       if((NextByte & 0x01)||((Position==0)&&(NextByte & 0x03)))
       {
          x--;
          Letter--;
          if(NoOfBlacks[NextByte] == 2)
             NextPosition = BlackPosition[NextByte & 0x07];
          goto FillVoid;
       }
    }

    if(x!=(LetterWidth-1) && ((Position==6)||(Position==7)))
    {
       NextByte= *(Letter+1);
       if((NextByte & 0x80)||((Position==7)&&(NextByte & 0xC0)))
       {
          x++;
          Letter++;
          if(NoOfBlacks[NextByte] == 2)
             NextPosition = BlackPosition[NextByte];
          goto FillVoid;
       }
    }
}
    /* When program has reached this point it       */
    /* indicates that no continuity has been        */
    /* found in the previous two checks. So         */
    /* assume that the lines were void due                    */
    /* to bit loss in stroke scanning and try the   */
    /* next line. But before that check if bottom   */
    /* edge of letter is reached, save              */
    /* coordinates and exit if yes.                 */

    /* Check this continuity only if LetterHeight is
    greater than 40. */

    VoidVoid=0;
    if((y<LetterHeight-2)&&(LetterHeight>30)&&\
          ((LetterWidth*8)>30))
    {
    VoidVoid=1;
    /* Check for bit just below the 'black' bit */
    /* under consideration */
```

```
        /* Make Letter point to the byte two lines below */
        /* the byte whose continuity is being checked */

        Letter += LetterWidth;

        /* Point to next line */

        y ++;

        /* Check for bit below the 'black' bit */
        /* under consideration */

        NextByte = *Letter;

        if (CurrentByte & NextByte) {
           if (NoOfBlacks[NextByte] == 2)
               NextPosition = Position;
                       goto FillVoidVoid;
        }

        if(Mask1 & NextByte) {
           if (NoOfBlacks[NextByte] == 2){
              if(Position > 3)
                 NextPosition = BlackPosition[NextByte & 0x0F];
              else
                 NextPosition = BlackPosition[NextByte];
           }
                        goto FillVoidVoid;
        }

        /* If the above check failed, check    */
        /* continuity in left and right bytes. */

        if(x!=0 && (Position==0))
        {
           NextByte= *(Letter-1);
                       if(NextByte & 0x01){
              x--;
              Letter--;
              if(NoOfBlacks[NextByte] == 2)
                       NextPosition = BlackPosition[NextByte & 0x03];
                                goto FillVoidVoid;
           }
        }

        if(x!=(LetterWidth-1) && (Position==7))
        {
           NextByte= *(Letter+1);
                       if(NextByte & 0x80){
              x++;
              Letter++;
              if(NoOfBlacks[NextByte] == 2)
                 NextPosition = BlackPosition[NextByte];
                                goto FillVoidVoid;
           }
        }
}
```

```
/* When program has reached this point it          */
            /* indicates that three consecutive lines have been    */
/* found without continuity, it means a            */
/* end of stroke, so reduce the y coordinate       */
/* by two, and go to save it.                      */

            if(VoidVoid)
                              y--;
if ((LetterHeight>30)&&((LetterWidth*8)>30))
                              y--;
if (y < LetterHeight-1)
                              y --;
goto EndStroke;

/* Fill out the coordinate for line found void */
/* but next line giving continuity */

            FillVoidVoid:
                  Strokes ++;
                  *Strokes = x * 8 + Position;

FillVoid:
                  Strokes ++;
                  *Strokes = x * 8 + Position;

   /* Save position of black */

SaveX:

   if( NoOfBlacks [NextByte] < 2)
      Position = BlackPosition [NextByte];
   else
      Position = NextPosition;

   /* Get byte to be whose continuity is to be */
   /* checked next */

   CurrentByte = 1 << (7 - Position);

   /* Zero out the black found in letter */

   *Letter = (*Letter) & ~(CurrentByte);

   /* Save x coordinate in Stroke array */

   Strokes ++;
   *Strokes = x * 8 + Position;

/*  fprintf(OutFile, "\n  X: %3d, Y: %3d", x * 8 + Position,
                  (LetterHeight-1)-y);
*/

   continue;

   /* For storing the end y coordinate of stroke */
   /* bring back pointer */
```

```
        EndStroke:

            Strokes = CurrentStroke;
            Strokes++;
            *Strokes = ((LetterHeight-1)-y);

            /* Update stroke counter */
            StrokeNumber ++;
            break;
      }
   }

   /* Actual value of StrokeNumber is one less  than the */
   /* current value, due to last the increment */

   StrokeNumber --;

   /* The previously allocated memory, Letter is no longer needed */
   /* Release it */

   Letter = StartOfLetter;
   free (Letter);

}


void GetLetterCode(void) {

   /* This function finds the letter code from the stroke    */
   /* coordinates found in the function StoreStrokes.        */
   /* Function FindStrokeCode is called to find the code for */
   /* a particular morph.                                    */

   unsigned char i, j=0;

   printf("\a\a");
   for (i = 0; i <= StrokeNumber; i ++)
   {
         /* Point to the start of stroke under consideration */
         Strokes = CurrentStroke =
                   StartOfStrokes + i * (LetterHeight + 2);

         /* Find code for the particular stroke */
         FindStrokeCode();

         /* Update LetterCode, but first check if StrokeCode */
         /* is '0' (rejected) */
         if(StrokeCode != '0') {
            LetterCode[j] = StrokeCode;
            StrokePercent[j] = MorphPercent;
            j++;
         }
   }
   CodeLength=j;

}
```

```
void FindStrokeCode() {

  /* This function finds the code for a particular stroke. The  */
  /* stroke may be a straight line or a curve. If the stroke is */
  /* a straight line, then it may be a horizontal line, a       */
  /* vertical line, a positive slope line, or a negative slope  */
  /* line. When a stroke is not a straight line, it may be a    */
  /* vertical concave curve or a vertical convex curve. A deci- */
  /* sion is taken whether a stroke is a straight line or a     */
  /* curve by a second order regration equation. Again the      */
  /* slope of a straight line is determined by a first order    */
  /* regration equation.                                        */

  int i, j;
  float x, y;
  float a, b, c, aa,bb,bx, cN;          /* Stroke curve     */
                                        /* coefficients and */
                                        /* ratio */
  float aNumerator, bNumerator, cNumerator,
        Denominator,deno;               /* Scratch variables */
                                        /* for calculating   */
                                        /* a, b, c */

  float y2;                             /* Scratch variable for */
                                        /* y square */
  float xSum, ySum, N, y2Sum,x2Sum, xySum,
        y3Sum, y2xSum, y4Sum;           /* Scratch variables.   */
                                        /* xSum denotes sum of  */
                                        /* x's. y2xSum denotes  */
                                        /* sum of y square into */
                                        /* x. N denotes total   */
                                        /* points in the curve  */

  /* Strokes now points to the third point of current stroke */
  /* memory location */

  Strokes ++;
  Strokes ++;

  /* Get starting y coordinate of stroke. */
  y = (float) (*CurrentStroke);

  /* Find total points in the curve */
  N = *(CurrentStroke) - *(CurrentStroke+1) + 1;

  /* Check if the stroke length is less than three or stroke */
  /* length is less than 6.5% of the Letter Height or stroke */
  /* length is less than 6.5% of the Letter Width.           */
  /* Reject if yes, assigning a value of zero to the         */
  /* StrokeCode, and going back to the calling program,      */
  /* GetLetterCode. */

  if(((!((N/(8*LetterWidth)*100)>6.5))&&\
        (!((N/LetterHeight*100)>6.5)))||(N<3))
  {
        StrokeCode = '0';
        return;
        }
```

```
/* Claculate the Stroke length in percent */

      MorphPercent = N/((float)LetterHeight)*100;

/* Initialize sums and ... */
xSum = ySum = y2Sum = x2Sum = xySum = y3Sum =
                      y2xSum = y3Sum = y4Sum = 0;
/* start a loop for finding the various sums */

for (i = 0; i < N; i ++)
{
/* Get current x coordinate. */

      x = (float) (*Strokes);

      xSum   = xSum   + x;
      ySum   = ySum   + y;
      y2     = y * y;
      x2Sum  = x2Sum  + x * x;
      y2Sum  = y2Sum  + y2;
      xySum  = xySum  + x * y;
      y3Sum  = y3Sum  + y2 * y;
      y2xSum = y2xSum + y2 * x;
      y4Sum  = y4Sum  + y2 * y2;

      /* Update pointer */
      Strokes ++;
      /* and y coordinate */
      y -= 1;
}
      /* Find c using relations for curve fitting.          */
      /* Coefficient a & b are not required for subsequnet */
      /* operations */

      /* X = a + bY + cY**2 */
/*
  aNumerator = xSum  * (y2Sum * y4Sum  -  y3Sum   * y3Sum)
             - ySum  * (xySum * y4Sum  -  y2xSum  * y3Sum)
             + y2Sum * (xySum * y3Sum  -  y2xSum  * y2Sum);

  bNumerator = N     * (xySum * y4Sum  -  y2xSum  * y3Sum)
             - xSum  * (ySum  * y4Sum  -  y2Sum   * y3Sum)
             + y2Sum * (ySum  * y2xSum -  y2Sum   * xySum);  */

  cNumerator = N     * (y2Sum * y2xSum -  y3Sum   * xySum)
             - ySum  * (ySum  * y2xSum -  y2Sum   * xySum)
             + xSum  * (ySum  * y3Sum  -  y2Sum   * y2Sum);

  Denominator= N     * (y2Sum * y4Sum -  y3Sum    * y3Sum)
             - ySum  * (ySum  * y4Sum -  y2Sum    * y3Sum)
             + y2Sum * (ySum  * y3Sum -  y2Sum    * y2Sum);

  if(Denominator) {
/*    aa = aNumerator/Denominator;
      bb = bNumerator/Denominator;  */
      c = cNumerator/Denominator;
      }
```

```
  else {

/*      fprintf(OutFile,"\n\nStroke no.= %d : Denominator = %10.5f",
                        KKStrokeNumber,Denominator); */

        StrokeCode = '3';
        return;
        }

        /* Check if a straight line. If a straight line, coefficient */
        /* of x square will be very small compared to that of x */

        cN = (fabs(c))*N;
        if (cN < 0.75)
        {
        /* X = a + bY */

           deno = N * y2Sum - ySum * ySum;
           if(deno) {
/*            a = (xSum * y2Sum - xySum * ySum)/deno; */
              b = (N * xySum - ySum * xSum)/deno;
           }
           else {
/*            fprintf(OutFile,"\n\nStroke no.= %d : Denominator = \
                                 %15.5f", KKStrokeNumber,deno);  */
              StrokeCode = '1';
              return;
           }

/*         fprintf(OutFile,"\n\nStroke no.= %d : N= %5.0f,\
                b= %10.5f,c= %10.5f",KKStrokeNumber,N,b,c); */

        /* Check the slope of straight line */
        /* If slope is in the range of 67.5 to 112.5 degree */
        /* it is taken as a vertical straight line. */
        /* tan 67.5 degree = 2.4142, tan 90 = Infinity */
        /* tan 157.5 = -2.4142 */

        if(fabs(b)<0.001){
           StrokeCode = '3';
           return;
        }
        else
           bx = 1/b;
           if ((bx > 2.4142) || (bx < -2.4142))
           {
              StrokeCode = '3';
              return;
           }

        /* If slope is in the range of -22.5 to 22.5 degree */
        /* or 157.5 to 180 degree it is taken as a horizontal */
        /* straight line. */
        /* tan 0 degree = 0.0, tan 22.5 = 0.4142, */
        /* tan 157.5 = -0.4142, tan 180 = 0.0 */

        if (((bx >= 0) && (bx <= 0.4142))
                        || ((bx <= 0) && (bx >= -0.4142)))
```

```
        {
           StrokeCode = '1';
           return;
        }

        /* If slope is in the range of 22.5 to 67.5 degree */
        /* it is taken as a positive slope straight line. */
        /* tan 22.5 degree = 0.4142, tan 67.5 = 2.4142 */

        if ((bx > 0.4142) && (bx <= 2.4142))
        {
           StrokeCode ='2';
           return;
        }
        /* If slope is in the range of 112.5 to 157.5 degree */
        /* it is taken as a negative slope straight line. */
        /* tan 112.5 degree = -2.4142, tan 157.5 = -0.4142 */

        if ((bx >= -2.4142) && (bx < -0.4142))
        {
           StrokeCode = '4';
           return;
        }
     }
     else
     {
     /* When not a straight line, the stroke is test the stroke */
     /* for vertical  concave/convex curve. */
     /* If c is less than zero it is a vertical convex */
     /* curve else it is a vertical concave curve */
        if (c < 0)
        {
/*      fprintf(OutFile,"\n\nStroke no.= %d : N = %5.0f,\
                bb= %10.5f c= %10.5f",KKStrokeNumber,N,bb,c); */

           StrokeCode = '6';
           return;
        }
        else
        {
/*      fprintf(OutFile,"\n\nStroke no.= %d : N = %5.0f,\
                bb= %10.5f c= %10.5f",KKStrokeNumber,N,bb,c); */

           StrokeCode = '5';
           return;
        }
     }
}
void SetGraphicsMode(void) {
        /* Set the graphics mode and clears the screen */

        int GraphDriver = DETECT, GraphMode;

         /* Set graphics mode */
         initgraph(&GraphDriver, &GraphMode, "c:\\turboc\\");
         /* Clear screen */
         cleardevice();
         }
```

```
void setprn(void)
 {
  fprintf(stdprn,"%c%c%c",27,'A',8);
 }

void resetprn(void)
 {
  fprintf(stdprn,"%c%c%c",27,'2',12);
 }

void initline( int gcol)
  {
   char a;
   fprintf(stdprn,"%c%c%c%c",27,'L',gcol,gcol>>8);
   }

void toPrint(int x1,int y1,int x2,int y2)
   {
    int yrange;
    int hlim,vlim,vext,klim,bcol,i,j,k;
    unsigned char pins;

    setprn();
    hlim = x2 - x1 + 1;
    yrange = ( y2 - y1 + 1 );
    vlim = yrange/8;
    vext = yrange%8;
    klim = 7;

    for ( i=0;i<=vlim;i++)
    {
       initline(hlim);
       for ( j=x1;j<=x2;j++)
       {
          if ( i == vlim ) klim = vext;
          pins = 0;
          for ( k=0; k<= klim ; k++)
             if  ( getpixel(j,i*8+k+y1) )
             pins += 1 << ( 7-k );
          fprintf(stdprn,"%c",pins);
       }
       fprintf(stdprn,"\n");
    }
    resetprn();
}

void DisplayLetter (unsigned XTop, unsigned YTop) {

  int x, y, j, k;                          /* Coordinate and */
                                           /* scratch variables */

  /* Each 8nth element in the array below indicates the start    */
  /* of binary form of any integer n, 0 to 255. For example      */
  /* element 16 (= 2 X 8) marks the beginning of,                */
  /* Decimal 2 = 0000 0010 = 0,0,0,0, 0,0,1,0                    */
  /* There are total 256 * 8 = 2048 elements in this array.      */
```

```
unsigned char BitMap [] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,
0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,
0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,1,
0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,1,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1,1,
0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,1,
0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,1,1,0,0,0,0,1,0,1,1,1,
0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,1,0,0,0,1,1,0,1,0,0,0,0,1,1,0,1,1,
0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,
0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,
0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,0,0,0,1,0,0,1,1,1,
0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,0,0,0,1,0,1,0,1,1,
0,0,1,0,1,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,0,0,0,1,0,1,1,1,1,
0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,1,0,0,1,1,
0,0,1,1,0,1,0,0,0,0,1,1,0,1,0,1,0,0,1,1,0,1,1,0,0,0,1,1,0,1,1,1,
0,0,1,1,1,0,0,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0,1,0,0,0,1,1,1,0,1,1,
0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,1,0,0,1,1,1,1,1,0,0,0,1,1,1,1,1,1,
0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,0,1,1,
0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,0,0,1,0,0,0,1,1,1,
0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,1,0,1,1,
0,1,0,0,1,1,0,0,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,0,0,1,1,1,1,
0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,0,0,1,0,1,0,0,1,1,
0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,0,0,1,0,1,0,1,1,1,
0,1,0,1,1,0,0,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,1,
0,1,0,1,1,1,0,0,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,1,1,
0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,0,1,1,0,0,0,1,0,0,1,1,0,0,0,1,1,
0,1,1,0,0,1,0,0,0,1,1,0,0,1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,
0,1,1,0,1,0,0,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,1,0,0,1,1,0,1,0,1,1,
0,1,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,1,
0,1,1,1,0,0,0,0,0,1,1,1,0,0,0,1,0,1,1,1,0,0,1,0,0,1,1,1,0,0,1,1,
0,1,1,1,0,1,0,0,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,0,0,1,1,1,0,1,1,1,
0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,1,0,1,1,1,1,0,1,0,0,1,1,1,1,0,1,1,
0,1,1,1,1,1,0,0,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,
1,0,0,0,0,1,0,0,1,0,0,0,0,1,0,1,1,0,0,0,0,1,1,0,1,0,0,0,0,1,1,1,
1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,1,1,
1,0,0,0,1,1,0,0,1,0,0,0,1,1,0,1,1,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,
1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,
1,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,
1,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,0,1,0,0,1,1,0,1,1,
1,0,0,1,1,1,0,0,1,0,0,1,1,1,0,1,1,0,0,1,1,1,1,0,1,0,0,1,1,1,1,1,
1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,
1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,
1,0,1,0,1,0,0,0,1,0,1,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,
1,0,1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,
1,0,1,1,0,0,0,0,1,0,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,0,1,1,0,0,1,1,
1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,1,1,0,1,1,0,1,1,0,1,0,1,1,0,1,1,1,
1,0,1,1,1,0,0,0,1,0,1,1,1,0,0,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,
1,0,1,1,1,1,0,0,0,1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,
1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,1,1,
1,1,0,0,0,1,0,0,1,1,0,0,0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,0,1,1,1,
1,1,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,1,0,1,1,
1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,1,1,0,0,1,1,1,0,1,1,0,0,1,1,1,1,
1,1,0,1,0,0,0,0,1,1,0,1,0,0,0,1,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,
1,1,0,1,0,1,0,0,1,1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,1,
1,1,0,1,1,0,0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,1,1,0,1,1,0,1,1,
1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,
1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1,0,1,1,1,0,0,0,1,1,
```

```
  1,1,1,0,0,1,0,0,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,0,1,1,1,0,0,1,1,1,
  1,1,1,0,1,0,0,0,1,1,1,0,1,0,0,1,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,1,
  1,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,1,
  1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,
  1,1,1,1,0,1,0,0,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,0,1,1,1,1,0,1,1,1,
  1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,
  1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1};

  SetGraphicsMode();

  Letter = StartOfLetter;                    /* Point to starting */
                                             /* location */

  for (y = 0; y < LetterHeight; y ++)
  {
     for (x = 0; x <LetterWidth; x ++)
     {

     /* Point to the proper element in array BitMap */
         k = (*Letter) * 8;
         /* Pixel position is eight times the byte number */
         j = x * 8;

         /* Display eight pixels, BitMap[] is 0 or 1 */
         putpixel(XTop + j,     YTop + y, BitMap [k   ]);
         putpixel(XTop + j + 1, YTop + y, BitMap [k +1]);
         putpixel(XTop + j + 2, YTop + y, BitMap [k +2]);
         putpixel(XTop + j + 3, YTop + y, BitMap [k +3]);
         putpixel(XTop + j + 4, YTop + y, BitMap [k +4]);
         putpixel(XTop + j + 5, YTop + y, BitMap [k +5]);
         putpixel(XTop + j + 6, YTop + y, BitMap [k +6]);
         putpixel(XTop + j + 7, YTop + y, BitMap [k +7]);

         Letter ++;
     }
  }
  getch();
/*  toprint(40,40,500,300); */
  closegraph();
}


void KeepInFile(void) {

  /* This function stores the character in 'Letter' in a file */
  /* specified by the user */

  int x, y, z;                    /* Coordinate variables */
  int k, t;                       /* Scratch variables */

  /* Each 8nth element in the array below indicates the start    */
  /* of binary form of any ineteger n, 0 to 255. For example     */
  /* element 16 (= 2 X 8) marks the beginning of,                */
  /* Decimal 2 = 0000 0010 = 0,0,0,0, 0,0,1,0                    */
  /* There are total 256 * 8 = 2048 elements in this array.      */

  unsigned char BitMap [] = {
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,
```

0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,
0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0,1,0,1,1,
0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,1,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1,1,
0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,1,
0,0,0,1,0,1,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,1,1,0,0,0,0,1,0,1,1,1,
0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,1,0,0,0,1,1,0,1,0,0,0,0,1,1,0,1,1,
0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,1,
0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,
0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,0,0,0,1,0,0,1,1,1,
0,0,1,0,1,0,0,0,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,0,0,0,1,0,1,0,1,1,
0,0,1,0,1,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,0,0,0,1,0,1,1,1,1,
0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,1,0,0,1,1,
0,0,1,1,0,1,0,0,0,0,1,1,0,1,0,1,0,0,1,1,0,1,1,0,0,0,1,1,0,1,1,1,
0,0,1,1,1,0,0,0,0,0,1,1,1,0,0,1,0,0,1,1,1,0,1,0,0,0,1,1,1,0,1,1,
0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,1,0,0,1,1,1,1,1,0,0,0,1,1,1,1,1,1,
0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,0,1,1,
0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,0,0,1,0,0,0,1,1,1,
0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,0,1,0,1,1,
0,1,0,0,1,1,0,0,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,0,0,1,1,1,1,
0,1,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,1,0,1,0,0,1,0,0,1,0,1,0,0,1,1,
0,1,0,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,0,0,1,0,1,0,1,1,1,
0,1,0,1,1,0,0,0,0,1,0,1,1,0,0,1,0,1,0,1,1,0,1,0,0,1,0,1,1,0,1,1,
0,1,0,1,1,1,0,0,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,1,1,
0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,1,0,1,1,0,0,0,1,0,0,1,1,0,0,0,1,1,
0,1,1,0,0,1,0,0,0,1,1,0,0,1,0,1,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,
0,1,1,0,1,0,0,0,0,1,1,0,1,0,0,1,0,1,1,0,1,0,1,0,0,1,1,0,1,0,1,1,
0,1,1,0,1,1,0,0,0,1,1,0,1,1,0,1,0,1,1,0,1,1,1,0,0,1,1,0,1,1,1,1,
0,1,1,1,0,0,0,0,0,1,1,1,0,0,0,1,0,1,1,1,0,0,1,0,0,1,1,1,0,0,1,1,
0,1,1,1,0,1,0,0,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,0,0,1,1,1,0,1,1,1,
0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,1,0,1,1,1,1,0,1,0,0,1,1,1,1,0,1,1,
0,1,1,1,1,1,0,0,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,
1,0,0,0,0,1,0,0,1,0,0,0,0,1,0,1,1,0,0,0,0,1,1,0,1,0,0,0,0,1,1,1,
1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,1,1,
1,0,0,0,1,1,0,0,1,0,0,0,1,1,0,1,1,0,0,0,1,1,1,0,1,0,0,0,1,1,1,1,
1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,1,
1,0,0,1,0,1,0,0,1,0,0,1,0,1,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,
1,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,0,1,0,0,1,1,0,1,1,
1,0,0,1,1,1,0,0,1,0,0,1,1,1,0,1,1,0,0,1,1,1,1,0,1,0,0,1,1,1,1,1,
1,0,1,0,0,0,0,0,1,0,1,0,0,0,0,1,1,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,
1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,0,1,0,1,0,0,1,1,1,
1,0,1,0,1,0,0,0,1,0,1,0,1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,1,
1,0,1,0,1,1,0,0,1,0,1,0,1,1,0,1,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,1,
1,0,1,1,0,0,0,0,1,0,1,1,0,0,0,1,1,0,1,1,0,0,1,0,1,1,0,0,0,1,1,1,
1,0,1,1,0,1,0,0,1,0,1,1,0,1,0,1,1,0,1,1,0,1,1,0,1,0,1,1,0,1,1,1,
1,0,1,1,1,0,0,0,1,0,1,1,1,0,0,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,1,
1,0,1,1,1,1,0,0,1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,0,1,0,1,1,1,1,1,1,
1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,0,1,1,0,0,0,0,1,1,
1,1,0,0,0,1,0,0,1,1,0,0,0,1,0,1,1,1,0,0,0,1,1,0,1,1,0,0,0,1,1,1,
1,1,0,0,1,0,0,0,1,1,0,0,1,0,0,1,1,1,0,0,1,0,1,0,1,1,0,0,1,0,1,1,
1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,1,1,1,0,0,1,1,1,0,1,1,0,0,1,1,1,1,
1,1,0,1,0,0,0,0,1,1,0,1,0,0,0,1,1,1,0,1,0,0,1,0,1,1,0,1,0,0,1,1,
1,1,0,1,0,1,0,0,1,1,0,1,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,1,1,1,
1,1,0,1,1,0,0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,0,1,1,0,1,1,0,1,1,
1,1,0,1,1,1,0,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,1,0,1,1,0,1,1,1,1,1,
1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1,0,1,1,1,0,0,0,1,1,
1,1,1,0,0,1,0,0,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,0,1,1,1,0,0,1,1,1,
1,1,1,0,1,0,0,0,1,1,1,0,1,0,0,1,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,1,

```
1,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,1,
1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1,0,0,1,0,1,1,1,1,0,0,1,1,
1,1,1,1,0,1,0,0,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,0,1,1,1,1,0,1,1,1,
1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,1,1,1,1,1,1,0,1,0,1,1,1,1,1,0,1,1,
1,1,1,1,1,1,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1
};

FILE *OutFile;
char FileName[31];              /* File name from user */
unsigned char TenCount, LineCount;
/* Get name of the output file from user */
clrscr();
printf("Enter file name in which to store the letter: ");
scanf("%s", FileName);

/* Open file, issue message if could not be opened */
if ((OutFile = fopen(FileName, "w")) == NULL)
{
   perror("\n\a\aSorry, file could not be created");
   exit(1);
}

/* Print dot commands for WS4 */
fprintf(OutFile, ".cw8 \n.lh4\n.op \n.pl%d \n.mt1 \n.mb1\
       \n.lm1 \n.rm%d \n",LetterHeight+5, LetterWidth * 9 + 3);

/* Print byte column numbers */
for (x = 0; x < LetterWidth; x ++)
{
    fprintf(OutFile, "%02d       ",x);
}

fprintf(OutFile, "\n");

Letter = StartOfLetter;

TenCount = 0;
LineCount = 0;
fprintf(OutFile, "%02d ", TenCount);
TenCount ++;
for (x = 0; x < LetterWidth; x ++)
{
   fprintf(OutFile, "---------");
}
fprintf(OutFile,"\n");

for (y = 0; y < LetterHeight; y ++)
{
   fprintf(OutFile,"  ");

   for (x = 0; x < LetterWidth; x ++)
   {
   /* Point to the proper element in array BitMap */
      k = (*Letter) * 8;

      /* Store the eight 1's or 0's, BitMap[] is 0 or 1 */
```

```
        for (t = 0; t <= 7; t ++)
        {
           if (BitMap [k + t])
           {
           /* Start bold in WS4 */
              putc (02, OutFile);
              /* Start double strike in WS4 */
              putc (04, OutFile);

              putc ('1', OutFile);

              /* End bold in WS4 */
              putc (02, OutFile);
              /* End double strike in WS4 */
              putc (04, OutFile);
           }

           else    putc ('0', OutFile);
        }

        fprintf(OutFile,"|");
        Letter ++;
     }
     fprintf(OutFile, "\n");            /* Carriage return at */
                                        /* the end of line */

     LineCount++;
     if (LineCount == 10)
     {
        LineCount = 0;
        fprintf(OutFile, "%02d ", TenCount);
        TenCount ++;
        for (z = 0; z < LetterWidth; z ++)
        {
           fprintf(OutFile, "---------");
        }
        fprintf(OutFile, "\n");
     }


  }

  /* Close the file */
  fclose (OutFile);
}
```

APPENDIX D

APPENDIX D

EXTERNAL FUNCTION RecognizeChar()

```
#include <stdio.h>
extern char LetterCode[10];
extern float StrokePercent[10];
extern int CodeLength;

void case2(void);
void case3(void);
void case4(void);
void case5(void);
void case6(void);

void RecognizeChar(void) {

switch(LetterCode[0]) {
  case '5': case5();
            break;
  case '4': case4();
            break;
  case '3': case3();
            break;
  case '2': case2();
            break;
  case '6': case6();
            break;
  }
}

void case2(void) {
  switch(LetterCode[1]) {
       case '2':switch(LetterCode[2]) {
            case '3':switch(LetterCode[3]) {
                  case '6': if(CodeLength==9&&(LetterCode[4]=='4'||LetterCode[4]=='5')
                        &&LetterCode[5]=='5'&&(LetterCode[6]=='3'||LetterCode[6]=='6')
                        &&(LetterCode[7]=='4'||LetterCode[7]=='5')\
                        &&(LetterCode[8]=='4'||LetterCode[8]=='6'))
                          printf("The Letter is: Niow.");
                 }
                 break;
            case '6':switch(LetterCode[3]) {
                   case '3':
                   case '4':
                   case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')
                       &&LetterCode[5]=='6')
                        printf("The Letter is: Ummaw.");
                 }
                 break;
            }
            break;
```

```
        case '4':switch(LetterCode[2]) {
             case '6':switch(LetterCode[3]) {
                    case '3':
                    case '4':
                    case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')\
                        &&LetterCode[5]=='6')
                         printf("The Letter is: Ummaw.");
                  }
                  break;
             }
             break;
        case '5':switch(LetterCode[2]) {
             case '3':switch(LetterCode[3]) {
                    case '6':if(CodeLength==9&&(LetterCode[4]=='4'||LetterCode[4]=='5')\
                         &&LetterCode[5]=='5'&&(LetterCode[6]=='3'||LetterCode[6]=='6')\
                         &&(LetterCode[7]=='4'||LetterCode[7]=='5')\
                         &&(LetterCode[8]=='4'||LetterCode[8]=='6'))
                           printf("The Letter is: Niow.");
                  }
             }
        }
}

void case3(void) {
    switch(LetterCode[1]) {
        case '2':switch(LetterCode[2]) {
             case '2':switch(LetterCode[3]) {
                    case '2':
                    case '3':
                    case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                         &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                         printf("The Letter is: Talibbashaw.");
                  }
                  break;
             case '3':if(CodeLength==3)
                    printf("The Letter is: Daow.");
                  break;
             case '4':switch(LetterCode[3]) {
                    case '2':
                    case '3':
                    case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                         &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                         printf("The Letter is: Talibbashaw.");

                  }
                  break;
             case '5':switch(LetterCode[3]) {
                    case '2':
                    case '3':
                    case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                         &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                         printf("The Letter is: Talibbashaw.");
                       break;
                    case '6':if(CodeLength==6&&LetterCode[4]=='4'&&(LetterCode[5]=='4'\
                          ||LetterCode[5]=='6'))
                          printf("The Letter is: Khaw.");
                  }
                  break;
```

```
        }
        break;
    case '3':switch(LetterCode[2]) {
        case '2':
        case '5':
        case '6':switch(LetterCode[3]) {
                case '2':
                case '5':if(CodeLength==8&&LetterCode[4]=='5'&&LetterCode[5]=='3'\
                     &&(LetterCode[6]=='4'||LetterCode[6]=='6')
                     &&(LetterCode[7]=='4'||LetterCode[7]=='6'))
                       printf("The Letter is: Rrie.");
            }
            break;
        }
        break;
    case '4':switch(LetterCode[2]) {
        case '2':switch(LetterCode[3]) {
                case '2':
                case '3':
                case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                     &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                     printf("The Letter is: Talibbashaw.");
                   break;
                case '6': if(CodeLength==4)
                     printf("The Letter is: Thaw.");
            }
            break;
        case '3':switch(LetterCode[3]) {
                case '3':
                case '5':if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
                    &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                     printf("The Letter is: Aaow.");
                   break;
                case '6':if(CodeLength==4)
                     printf("The Letter is: Thaw.");
                   if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
                    &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                     printf("The Letter is: Aaow.");
            }
            break;
        case '4':switch(LetterCode[3]) {
                case '2':
                case '3':
                case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                     &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                     printf("The Letter is: Talibbashaw.");
                   break;
            }
            break;
        case '5':switch(LetterCode[3]) {
                case '2':if(CodeLength==6&&LetterCode[4]=='3'\
                     &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                     printf("The Letter is: Talibbashaw.");
                   break;
                case '3':if(CodeLength==6&&LetterCode[4]=='3'\
                     &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                     printf("The Letter is: Talibbashaw.");
                   if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
```

```
                &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                 printf("The Letter is: Aaow.");
               if(CodeLength==6 &&(LetterCode[4]=='4'||LetterCode[4]=='6')\
                 &&(LetterCode[5]=='4'||LetterCode[5]=='6'))
                 printf("The Letter is: Jhaw.");
               if(CodeLength==7&&LetterCode[4]=='4'\
                &&LetterCode[5]=='6'&& LetterCode[6]=='4')
                 printf("The Letter is: Aaow.");
               break;
           case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                 &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                 printf("The Letter is: Talibbashaw.");
               if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
                &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                 printf("The Letter is: Aaow.");
               break;
           case '6':if(CodeLength==4)
                 printf("The Letter is: Thaw.");
               if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
                &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                 printf("The Letter is: Aaow.");
         }
         break;

     }
     break;
case '5':switch(LetterCode[2]) {
     case '2':switch(LetterCode[3]) {
           case '2':
           case '3':if(CodeLength==6&&LetterCode[4]=='3'\
                 &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                 printf("The Letter is: Talibbashaw.");
               if(CodeLength==5&&(LetterCode[4]=='2'||LetterCode[4]=='3'))
                 printf("The Letter is: Paw.");
                 break;
           case '5':if(CodeLength==6&&LetterCode[4]=='3'\
                 &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                 printf("The Letter is: Talibbashaw.");
         }
         break;
     case '3':switch(LetterCode[3]) {
           case '2': if(CodeLength==4)
                 printf("The Letter is: ModinanNaw.");
                 break;
           case '3': if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')
               &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                 printf("The Letter is: Aaow.");
                 break;
           case '5': if(CodeLength==4)
                 if(StrokePercent[3]>25)
                   printf("The Letter is: Gaw.");
                 else
                   printf("The Letter is: ModinanNaw.");
                  if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
               &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
                 printf("The Letter is: Aaow.");
                 break;
           case '6': if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')
```

```
            &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
             printf("The Letter is: Aaow.");
        }
        break;
case '4':switch(LetterCode[3]) {
        case '2':
        case '3':
        case '5': if(CodeLength==6&&LetterCode[4]=='3'\
             &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
             printf("The Letter is: Talibbashaw.");
        }
        break;
case '5':switch(LetterCode[3]) {
        case '2': if(CodeLength==6&&LetterCode[4]=='3'\
             &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
             printf("The Letter is: Talibbashaw.");
             break;
        case '3': switch(LetterCode[4]) {
             case '3':if(CodeLength==6 &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
                  printf("The Letter is: Talibbashaw.");
                if(CodeLength==8&&LetterCode[5]=='4'&& LetterCode[6]=='6'\
                  &&LetterCode[7]=='4')
                  printf("The Letter is: Aaow.");
                break;
             case '5':if(CodeLength==8&&LetterCode[5]=='4'&& LetterCode[6]=='6'\
                  &&LetterCode[7]=='4')
                  printf("The Letter is: Aaow.");
                break;
             case '4':
             case '6':if(CodeLength==6 &&(LetterCode[5]=='4'||LetterCode[5]=='6'))
                  printf("The Letter is: Jhaw.");
                break;
            }
            break;
        case '5': if(CodeLength==6&&LetterCode[4]=='3'\
             &&(LetterCode[5]=='4'||LetterCode[5]=='5'))
             printf("The Letter is: Talibbashaw.");
            if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
            &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
             printf("The Letter is: Aaow.");
            break;
        case '6': if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='5')\
            &&LetterCode[5]=='4'&& LetterCode[6]=='6'&& LetterCode[7]=='4')
             printf("The Letter is: Aaow.");
             break;

        }
        break;
case '6':switch(LetterCode[3]) {
        case '2':
        case '3': if(CodeLength==5&&(LetterCode[4]=='2'||LetterCode[4]=='3'))
             printf("The Letter is: Paw.");
             break;
        case '4': if(CodeLength==5&&(LetterCode[4]=='4'||LetterCode[4]=='6'))
             printf("The Letter is: Thaow.");
             break;
        }
        break;
```

```
            }
            break;
        case '6': if(CodeLength==5&&LetterCode[2]=='4'&&LetterCode[3]=='3'\
             &&(LetterCode[4]=='4'||LetterCode[4]=='6'))
             printf("The Letter is: Dhaow.");
            break;

        }
    }

void case4(void) {
    switch(LetterCode[1]) {
        case '2':switch(LetterCode[2]) {
            case '2':
            case '3':
            case '4':
            case '5': if(CodeLength==3)
              printf("The Letter is: ChandraBindoo.");
              break;
            case '6':switch(LetterCode[3]) {
                  case '3':
                  case '4':
                  case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')\
                      &&LetterCode[5]=='6')
                       printf("The Letter is: Ummaw.");
                       break;
                }
                break;
            }
            break;
        case '3':switch(LetterCode[2]) {
            case '2':if(CodeLength==3)
                    printf("The Letter is: ChandraBindoo.");
                switch(LetterCode[3]) {
                  case '2':
                  case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                       ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
                       ||LetterCode[5]=='6')&&LetterCode[6]=='5')
                     printf("The Letter is: Law.");
                     break;
                  case '3':if(CodeLength==4)
                       printf("The Letter is: Daow.");
                       break;
                }
                break;
            case '3':if(CodeLength==3)
                 if(StrokePercent[0]<20)
                   printf("The Letter is: Ddhaw.");
                 else
                   printf("The Letter is: ChandraBindoo.");
                switch(LetterCode[3]) {
                  case '2':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='5')
                     printf("The Letter is: DhoRaw.");
                     break;
                  case '4':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
                     printf("The Letter is: Aow.");
                     break;
```

```
        case '5':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
            if(CodeLength==4)
              printf("The Letter is: DhoShinnaRaw.");
            break;
      }
      break;
case '4':if(CodeLength==3)
         printf("The Letter is: ChandraBindoo.");
      switch(LetterCode[3]) {
        case '2':if(CodeLength==5&&LetterCode[4]=='6')
            printf("The Letter is: Ddaw.");
            if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='5')
            printf("The Letter is: DhoRaw.");
            break;
        case '4':
        case '5':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
            break;
      }
      break;
case '5':if(CodeLength==3)
         printf("The Letter is: ChandraBindoo.");
      switch(LetterCode[3]) {
        case '2':if(CodeLength==4)
            printf("The Letter is: Ddhaw.");
            if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
              ||LetterCode[5]=='6')&&LetterCode[6]=='5')
              printf("The Letter is: Law.");
            if(CodeLength==5&& LetterCode[4]=='5')
              printf("The Letter is: DoshinnaRaw.");
            break;
        case '3':if(CodeLength==5&&(LetterCode[4]=='4'||LetterCode[4]=='6'))
              printf("The Letter is: ModinasShaw.");
            if(CodeLength==6&&LetterCode[4]=='5'\
             &&(LetterCode[5]=='4'||LetterCode[5]=='6'))
              printf("The Letter is: Phaw.");
            break;
        case '4':if(CodeLength==4)
               if(StrokePercent[0]<20&&StrokePercent[1]<50)
                printf("The Letter is: DonTanNaw.");
               else if(StrokePercent[0]>27)
                printf("The Letter is: Ghaw.");
               else if(StrokePercent[0]<27&&StrokePercent[1]>50)
                printf("The Letter is: OnTastaJaw.");
            if(CodeLength==5&& LetterCode[4]=='5')
              printf("The Letter is: OntasTaw.");
            if(CodeLength==6&&LetterCode[4]=='6'&& LetterCode[5]=='4')
              printf("The Letter is: Aow.");
            break;
        case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
              ||LetterCode[5]=='6')&&LetterCode[6]=='5')
              printf("The Letter is: Law.");
            if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
              printf("The Letter is: Aow.");
            break;
```

```
            case '6':if(CodeLength==4)
                  if(StrokePercent[1]>75)
                    printf("The Letter is: Ddhaw.");
                  else if(StrokePercent[0]>27)
                    printf("The Letter is: Ghaw.");
                  else if(StrokePercent[0]<27&&StrokePercent[1]<75)
                    printf("The Letter is: OnTastaJaw.");
                if(CodeLength==5&& LetterCode[4]=='5')
                  if(StrokePercent[0]>15&&StrokePercent[2]>35)
                    printf("The Letter is: OntasTaw.");
                   else
                    printf("The Letter is: DhoShinnaRaw.");
                if(CodeLength==6&&LetterCode[4]=='5'\
                 &&(LetterCode[5]=='4'||LetterCode[5]=='6'))
                  printf("The Letter is: Phaw.");
            }
            break;
      case '6':if(CodeLength==3)
                printf("The Letter is: Chaw.");
            switch(LetterCode[3]) {
              case '3':if(CodeLength==5&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                    ||LetterCode[4]=='5'))
                    printf("The Letter is: DontaShaw.");
                  break;
              case '5':if(CodeLength==5&&(LetterCode[4]=='3'||LetterCode[4]=='5'))
                    printf("The Letter is: DontaShaw.");
                  if(CodeLength==5&&LetterCode[4]=='4')
                    printf("The Letter is: Maow");
                  break;
              case '6':if(CodeLength==6&&LetterCode[4]=='4'&&(LetterCode[5]=='4'\
                    ||LetterCode[5]=='6'))
                    printf("The Letter is: Chhaw.");
                  break;
            }
            break;
      }
      break;
 case '4':switch(LetterCode[2]) {
      case '2':
      case '3':
      case '4':
      case '5': if(CodeLength==3)
                printf("The Letter is: ChandraBindoo.");
                if(CodeLength==5&&LetterCode[3]=='2'&&LetterCode[4]=='6')
                printf("The Letter is: Bhaw.");
                if(CodeLength==4&&LetterCode[3]=='6')
                printf("The Letter is: Taow.");
                break;
      case '6':switch(LetterCode[3]) {
                 case '3':
                 case '4':
                 case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')\
                     &&LetterCode[5]=='6')
                      printf("The Letter is: Ummaw.");
                      break;
            }
            break;
      }
      break;
```

```
case '5':switch(LetterCode[2]) {
     case '2': if(CodeLength==3)
           printf("The Letter is: ChandraBindoo.");
     case '3':if(CodeLength==3)
           if(StrokePercent[0]<20)
             printf("The Letter is: Ddhaw.");
           else
             printf("The Letter is: ChandraBindoo.");
          switch(LetterCode[3]) {
            case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                 ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                 printf("The Letter is: Jaw.");
               if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                 ||LetterCode[5]=='6'))
                 printf("The Letter is: Kaw.");
               break;
           case '4':if(CodeLength==4)
                 printf("The Letter is: Baw.");
                if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
                  ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
                  ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
                  printf("The Letter is: Jaw.");
                if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                  ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                  printf("The Letter is: Jaw.");
                if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                  ||LetterCode[5]=='6'))
                  printf("The Letter is: Kaw.");
                if(CodeLength==5&&LetterCode[4]=='5')
                  printf("The Letter is: Raw.");
                if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
                  printf("The Letter is: Aow.");
                break;
           case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
                  ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
                  ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
                  printf("The Letter is: Jaw.");
                if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
                  printf("The Letter is: Aow.");
                break;
            case '6':if(CodeLength==4)
                  printf("The Letter is: Baw.");
                if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                  ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                  printf("The Letter is: Jaw.");
                if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                  ||LetterCode[5]=='6'))
                  printf("The Letter is: Kaw.");
                if(CodeLength==5&&LetterCode[4]=='5')
                  printf("The Letter is: Raw.");
               break;
          }
          break;
     case '4': if(CodeLength==3)
           printf("The Letter is: ChandraBindoo.");
          switch(LetterCode[3]) {
            case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                 ||LetterCode[4]=='6')&&LetterCode[5]=='6')
```

```
            printf("The Letter is: Jaw.");
          break;
      case '4':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
          break;
      }
      break;
case '5':if(CodeLength==3)
      printf("The Letter is: ChandraBindoo.");
      switch(LetterCode[3]) {
        case '2':if(CodeLength==4)
            printf("The Letter is: Ddhaw.");
          break;
        case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
          break;
        case '4':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
            &&LetterCode[5]=='4')
            printf("The Letter is: Haw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Haw.");
          break;
        case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '6':if(CodeLength==4)
            printf("The Letter is: Ddhaw.");
          if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
```

```
                ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                printf("The Letter is: Jaw.");
              if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                printf("The Letter is: Haw.");
            break;
        }
        break;
    case '6':if(CodeLength==3)
         printf("The Letter is: Chaw.");
        switch(LetterCode[3]) {
          case '3':if(CodeLength==6&& LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                ||LetterCode[5]=='6'))
                printf("The Letter is: Kaw.");
            break;
          case '4':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                printf("The Letter is: Haw.");
              if(CodeLength==6&& LetterCode[4]=='5'&&(LetterCode[5]=='4'\
               ||LetterCode[5]=='6'))
               printf("The Letter is: Kaw.");
              if(CodeLength==5&&(LetterCode[4]=='6'||LetterCode[4]=='4'))
                printf("The Letter is: Haw.");
              break;
          case '5':if(CodeLength==4)
                printf("The Letter is: Haw.");
              break;
         case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                printf("The Letter is: Haw.");
              if(CodeLength==6&& LetterCode[4]=='5'&&(LetterCode[5]=='4'\
               ||LetterCode[5]=='6'))
               printf("The Letter is: Kaw.");
            break;
        }
        break;
    }
    break;
case '6':switch(LetterCode[2]) {
    case '2':
    case '3': if(CodeLength==3)
         printf("The Letter is: ChandraBindoo.");
    case '4': if(CodeLength==3)
         printf("The Letter is: ChandraBindoo.");
        switch(LetterCode[3]) {
          case '3':if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
              &&(LetterCode[5]=='3'||LetterCode[5]=='4')\
              &&LetterCode[6]=='2'&&LetterCode[7]=='6')
              printf("The Letter is: DirghaOow.");
            break;
        }
        break;
    case '5': if(CodeLength==3)
         printf("The Letter is: ChandraBindoo.");
        switch(LetterCode[3]) {
          case '3':if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
              &&(LetterCode[5]=='3'||LetterCode[5]=='4')\
              &&LetterCode[6]=='2'&&LetterCode[7]=='6')
```

```
                    printf("The Letter is: DirghaOow.");
                  if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                    ||LetterCode[4]=='6')&&LetterCode[5]=='5'\
                    &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
                    printf("The Letter is: Kaw.");
                  if(CodeLength==5&&(LetterCode[4]=='6'||LetterCode[4]=='4'))
                    printf("The Letter is: Baw.");
                  if(CodeLength==6&&(LetterCode[4]=='6'||LetterCode[4]=='4')\
                    &&LetterCode[5]=='5')
                    printf("The Letter is: Raw.");
                  break;
             case '4':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                      ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                      printf("The Letter is: Haw.");
                    break;
             case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                      ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                      printf("The Letter is: Haw.");
                    if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                     ||LetterCode[4]=='6')&&LetterCode[5]=='5'\
                     &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
                     printf("The Letter is: Kaw.");
                  break;
                }
                break;
          case '6':switch(LetterCode[3]) {
                 case '4':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                          ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                          printf("The Letter is: Haw.");
                  break;
                 case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                          ||LetterCode[4]=='6')&& LetterCode[5]=='4')
                          printf("The Letter is: Haw.");
                  break;
                }
                break;


          }
          break;
  }
}
void case5(void) {
    switch(LetterCode[1]) {
       case '2':switch(LetterCode[2]) {
            case '2':
            case '4':
            case '3': if(CodeLength==3)
                  printf("The Letter is: ChandraBindoo.");
                  switch(LetterCode[3]) {
                  case '6': if(CodeLength==9&&(LetterCode[4]=='4'||LetterCode[4]=='5'||Lette
                        &&LetterCode[5]=='5'&&(LetterCode[6]=='3'||LetterCode[6]=='6')\
                        &&(LetterCode[7]=='4'||LetterCode[7]=='5')\
                        &&(LetterCode[8]=='4'||LetterCode[8]=='6'))
                          printf("The Letter is: Niow.");
                  }
                  break;
            case '5': if(CodeLength==3)
```

```
            printf("The Letter is: ChandraBindoo.");
            if(CodeLength==4&&(LetterCode[3]=='2'||LetterCode[3]=='6'))
            printf("The Letter is: BissoGhaw.");
            break;
      case '6':switch(LetterCode[3]) {
             case '3':
             case '4':
             case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')\
                 &&LetterCode[5]=='6')
                  printf("The Letter is: Ummaw.");
                  break;
            }
            break;
      }
      break;
 case '3':switch(LetterCode[2]) {
      case '2':if(CodeLength==3)
               printf("The Letter is: ChandraBindoo.");
            switch(LetterCode[3]) {
              case '2':
              case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                   ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
                   ||LetterCode[5]=='6')&&LetterCode[6]=='5')
                 printf("The Letter is: Law.");
                 break;
            }
            break;
      case '3':if(CodeLength==3)
             if(StrokePercent[0]<20)
               printf("The Letter is: Ddhaw.");
             else
               printf("The Letter is: ChandraBindoo.");
            switch(LetterCode[3]) {
              case '4':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
                 printf("The Letter is: Aow.");
                 break;
              case '5':if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
                 printf("The Letter is: Aow.");
                 if(CodeLength==4)
                   printf("The Letter is: DhoShinnaRaw.");
                 break;
            }
            break;
      case '4':if(CodeLength==3)
               printf("The Letter is: ChandraBindoo.");
            switch(LetterCode[3]) {
              case '2':if(CodeLength==5&&LetterCode[4]=='6')
                   printf("The Letter is: Ddaw.");
              case '4':if(CodeLength==4)
                   printf("The Letter is: A.");
                 if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
                   printf("The Letter is: Aow.");
                 break;
              case '5':if(CodeLength==5&& LetterCode[4]=='6')
                   printf("The Letter is: O.");
                 if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
                   printf("The Letter is: Aow.");
                 if(CodeLength==7&& LetterCode[4]=='6'&&LetterCode[5]=='4'\
```

```
            &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
            printf("The Letter is: E.");
          if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '6':if(CodeLength==4)
            printf("The Letter is: A.");
    }
    break;
case '5':if(CodeLength==3)
        printf("The Letter is: ChandraBindoo.");
    switch(LetterCode[3]) {
      case '2':if(CodeLength==4)
          printf("The Letter is: Ddhaw.");
          if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
            ||LetterCode[5]=='6')&&LetterCode[6]=='5')
            printf("The Letter is: Law.");
          break;
      case '4':if(CodeLength==4)
             if(StrokePercent[0]>20)
              printf("The Letter is: A.");
             else
              printf("The Letter is: DontanNaw.");
          if(CodeLength==5&& LetterCode[4]=='5')
            printf("The Letter is: OntasTaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&& LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='4'\
            ||LetterCode[5]=='6')&&LetterCode[6]=='5')
            printf("The Letter is: Law.");
          if(CodeLength==6&& LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          if(CodeLength==7&&LetterCode[4]=='6'&&LetterCode[5]=='4'\
            &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
            printf("The Letter is: E.");
          break;
      case '6':if(CodeLength==4)
             if(StrokePercent[0]>20)
              printf("The Letter is: A.");
             else
              printf("The Letter is: Ddhaw.");
          if(CodeLength==5&& LetterCode[4]=='5')
            printf("The Letter is: Ontasta.");
          break;
    }
    break;
case '6':if(CodeLength==3)
        printf("The Letter is: Chaw.");
    if(CodeLength==5&&LetterCode[3]=='5'&&LetterCode[4]=='4')
        printf("The Letter is: Maow.");
    break;
}
break;
```

```
case '4':switch(LetterCode[2]) {
     case '2': if(CodeLength==3)
               printf("The Letter is: ChandraBindoo.");
               if(CodeLength==4&&LetterCode[3]=='6')
               printf("The Letter is: Thaw.");
               break;
     case '3': if(CodeLength==3)
               printf("The Letter is: ChandraBindoo.");
               if(CodeLength==4&&LetterCode[3]=='6')
               printf("The Letter is: Thaw.");
               break;
     case '4':
     case '5': if(CodeLength==3)
               printf("The Letter is: ChandraBindoo.");
               if(CodeLength==5&&LetterCode[3]=='2'&&LetterCode[4]=='6')
               printf("The Letter is: Bhaw.");
               if(CodeLength==4&&LetterCode[3]=='6')
                if(StrokePercent[0]>20)
                 printf("The Letter is: Thaw.");
                else
                 printf("The Letter is: Taow.");
               break;
     case '6':switch(LetterCode[3]) {
                case '3':
                case '4':
                case '5':if(CodeLength==6&&(LetterCode[4]=='2'||LetterCode[4]=='6')\
                     &&LetterCode[5]=='6')
                      printf("The Letter is: Ummaw.");
                      break;
              }
              break;
     }
     break;
case '5':switch(LetterCode[2]) {
     case '2':if(CodeLength==3)
              printf("The Letter is: ChandraBindoo.");
     case '3':if(CodeLength==3)
              if(StrokePercent[0]<20)
                printf("The Letter is: Ddhaw.");
              else
                printf("The Letter is: ChandraBindoo.");
             switch(LetterCode[3]) {
               case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                    ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                    printf("The Letter is: Jaw.");
                  if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                    ||LetterCode[5]=='6'))
                    printf("The Letter is: Kaw.");
                  break;
              case '4':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
                     ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
                     ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
                     printf("The Letter is: Jaw.");
                    if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                     ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                     printf("The Letter is: Jaw.");
                    if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
                     ||LetterCode[5]=='6'))
```

```
            printf("The Letter is: Kaw.");
          if(CodeLength==5&&LetterCode[4]=='5')
            printf("The Letter is: Raw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
           ||LetterCode[5]=='6'))
            printf("The Letter is: Dhaow.");
          break;
       case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
         if(CodeLength==6&&LetterCode[4]=='5'&&(LetterCode[5]=='4'\
            ||LetterCode[5]=='6'))
            printf("The Letter is: Kaw.");
         if(CodeLength==5&&LetterCode[4]=='5')
            printf("The Letter is: Raw.");
         if(CodeLength==9&&(LetterCode[4]=='4'||LetterCode[4]=='5'||LetterCod
           &&LetterCode[5]=='5'&&(LetterCode[6]=='3'||LetterCode[6]=='6')\
           &&(LetterCode[7]=='4'||LetterCode[7]=='5')\
           &&(LetterCode[8]=='4'||LetterCode[8]=='6'))
             printf("The Letter is: Niow.");
         break;
     }
     break;
case '4': if(CodeLength==3)
      printf("The Letter is: ChandraBindoo.");
     switch(LetterCode[3]) {
       case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
         break;
      case '4':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          break;
      case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
            ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
            ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
            printf("The Letter is: Jaw.");
          if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
            printf("The Letter is: Aow.");
          if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
           ||LetterCode[5]=='6'))
            printf("The Letter is: Dhaow.");
```

```
            break;
        case '6':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                ||LetterCode[4]=='6')&&LetterCode[5]=='6')
                printf("The Letter is: Jaw.");
            break;
      }
      break;
case '5':if(CodeLength==3)
       printf("The Letter is: ChandraBindoo.");
      switch(LetterCode[3]) {
        case '2':if(CodeLength==4)
              printf("The Letter is: Ddhaw.");
            break;
        case '3':if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&&LetterCode[5]=='6')
              printf("The Letter is: Jaw.");
            break;
        case '4':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
              ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
              ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
              printf("The Letter is: Jaw.");
            if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&&LetterCode[5]=='6')
              printf("The Letter is: Jaw.");
            if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
              printf("The Letter is: Aow.");
            break;
        case '5':if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='2'\
              ||LetterCode[4]=='6')&&(LetterCode[5]=='3'||LetterCode[5]=='2'\
              ||LetterCode[5]=='6'||LetterCode[5]=='4')&&LetterCode[6]=='6')
              printf("The Letter is: Jaw.");
            if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
             ||LetterCode[5]=='6'))
              printf("The Letter is: Dhaow.");
            if(CodeLength==6&&LetterCode[4]=='6'&&LetterCode[5]=='4')
              printf("The Letter is: Aow.");
            break;
       case '6':if(CodeLength==4)
              printf("The Letter is: Ddhaw.");
            if(CodeLength==6&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&&LetterCode[5]=='6')
              printf("The Letter is: Jaw.");
           break;
      }
      break;
case '6':if(CodeLength==3)
       printf("The Letter is: Chaw.");
      switch(LetterCode[3]) {
        case '3':
        case '4':
        case '6':
           if(CodeLength==6&& LetterCode[4]=='5'&&(LetterCode[5]=='4'\
             ||LetterCode[5]=='6'))
             printf("The Letter is: Kaw.");
           break;
      }
      break;
}
```

```
     break;
case '6':switch(LetterCode[2]) {
     case '2': if(CodeLength==3)
           printf("The Letter is: ChandraBindoo.");
     case '3': if(CodeLength==3)
            if(StrokePercent[1]<20)
             printf("The Letter is: KhondaTaw.");
           else
             printf("The Letter is: ChandraBindoo.");
           if(CodeLength==4&&(LetterCode[3]=='4'||LetterCode[3]=='6'))
           printf("The Letter is: KhondaTaw.");
           if(CodeLength==6&&LetterCode[3]=='5'&&LetterCode[4]=='3'\
             &&(LetterCode[5]=='4'||LetterCode[5]=='6'))
                 printf("The Letter is: Dhaow.");
           break;
     case '4':if(CodeLength==3)
           if(StrokePercent[1]<20)
             printf("The Letter is: KhondaTaw.");
           else
             printf("The Letter is: ChandraBindoo.");
         switch(LetterCode[3]) {
           case '2':if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
                &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
                &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
                printf("The Letter is: DirghaE.");
              break;
           case '3':if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
                &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
                &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
                printf("The Letter is: DirghaE.");
              if(CodeLength==5&&(LetterCode[4]=='3'||LetterCode[4]=='4'))
                printf("The Letter is: Taw.");
              if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
                &&LetterCode[5]=='2'&&LetterCode[6]=='6')
                printf("The Letter is: RasshoOow.");
              if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
                &&(LetterCode[5]=='3'||LetterCode[5]=='4')\
                &&LetterCode[6]=='2'&&LetterCode[7]=='6')
                printf("The Letter is: DirghaOow.");
              if(CodeLength==6&&LetterCode[4]=='5'&&\
                (LetterCode[5]=='6'||LetterCode[5]=='2'))
                printf("The Letter is: Taw.");
              break;
           case '4':if(CodeLength==4)
                printf("The Letter is: KhondaTaw.");
           case '5':if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
                &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
                &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
                printf("The Letter is: DirghaE.");
              if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
                ||LetterCode[5]=='6'))
                 printf("The Letter is: Dhaow.");
              if(CodeLength==5&&LetterCode[4]=='6')
                printf("The Letter is: O.");
              if(CodeLength==7&&LetterCode[4]=='6'&&LetterCode[5]=='4'\
                &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
                printf("The Letter is: E.");
              if(CodeLength==6&&LetterCode[4]=='6'&&\
```

```
            (LetterCode[5]=='4'||LetterCode[5]=='5'))
            printf("The Letter is: E.");
          break;
      }
      break;
case '5':if(CodeLength==3)
       if(StrokePercent[0]>60)
         printf("The Letter is: KhondaTaw.");
       else
         printf("The Letter is: UnoShar.");
      switch(LetterCode[3]) {
        case '2':if(CodeLength==4)
            printf("The Letter is: BisoGhaw.");
          if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
            &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
            &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
            printf("The Letter is: DirghaE.");
          break;
        case '3':if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
            &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
            &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
            printf("The Letter is: DirghaE.");
          if(CodeLength==5&&(LetterCode[4]=='3'||LetterCode[4]=='4'))
            printf("The Letter is: Taw.");
          if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
            &&LetterCode[5]=='2'&&LetterCode[6]=='6')
            printf("The Letter is: RasshoOow.");
          if(CodeLength==8&&(LetterCode[4]=='3'||LetterCode[4]=='4')\
            &&(LetterCode[5]=='3'||LetterCode[5]=='4')\
            &&LetterCode[6]=='2'&&LetterCode[7]=='6')
            printf("The Letter is: DirghaOow.");
          if(CodeLength==6&&LetterCode[4]=='5'&&\
            (LetterCode[5]=='6'||LetterCode[5]=='2'))
            printf("The Letter is: Taw.");
          if(CodeLength==6&&(LetterCode[4]=='5'||\
            LetterCode[4]=='3')&& LetterCode[5]=='4')
            printf("The Letter is: Ooi.");
          if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
            ||LetterCode[4]=='6')&&LetterCode[5]=='5'\
            &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
            printf("The Letter is: Kaw.");
          if(CodeLength==6&&(LetterCode[4]=='6'||LetterCode[4]=='4')\
            &&LetterCode[5]=='5')
            printf("The Letter is: Raw.");
          break;
       case '4': if(CodeLength==4)
             printf("The Letter is: KhondaTaw.");
            if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
              ||LetterCode[4]=='6')&& LetterCode[5]=='5'&& LetterCode[6]=='6')
             printf("The Letter is: Oou.");
           break;
       case '5':if(CodeLength==8&&LetterCode[4]=='2'&&LetterCode[5]=='6'\
            &&(LetterCode[6]=='3'||LetterCode[6]=='4')\
            &&(LetterCode[7]=='4'||LetterCode[7]=='5'||LetterCode[7]=='3'))
            printf("The Letter is: DirghaE.");
          if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
            ||LetterCode[5]=='6'))
             printf("The Letter is: Dhaow.");
```

```
                    if(CodeLength==7&&LetterCode[4]=='6'&&LetterCode[5]=='4'\
                      &&(LetterCode[6]=='4'||LetterCode[6]=='6'))
                      printf("The Letter is: E.");
                    break;
                case '6':if(CodeLength==4)
                      if(StrokePercent[0]<55)
                        printf("The Letter is: BishoGaw.");
                      else
                        printf("The Letter is: KhandaTaw.");
                     if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                      ||LetterCode[4]=='6')&&LetterCode[5]=='5'\
                      &&(LetterCode[6]=='4'))
                      printf("The Letter is: Kaw.");
                    if(CodeLength==7&&(LetterCode[4]=='3'||LetterCode[4]=='4'\
                      ||LetterCode[4]=='6')&&LetterCode[5]=='5'\
                      &&(LetterCode[6]=='6'))
                      if(StrokePercent[0]<17.5)
                        printf("The Letter is: Kaw.");
                      else
                        printf("The Letter is: Oou.");
                    break;
                }
                break;


            }
            break;
  }
}
void case6(void) {
    switch(LetterCode[1]) {
       case '5':
       case '6':switch(LetterCode[2]) {
            case '3':
            case '4':
            case '5':switch(LetterCode[3]) {
                  case '5':if(CodeLength==6&&LetterCode[4]=='3'&&(LetterCode[5]=='4'\
                        ||LetterCode[5]=='6'))
                         printf("The Letter is: Dhaow.");
                  }
                  break;
            }
            break;
       }
}
```

বাংলাদেশ প্রকৌশল বিশ্ববিদ্যালয়
নং 85057
তাং 7-10-92
লাইব্রেরী, ঢাকা