DISCRETE MATHEMATICAL STRUCTURES WITH APPLICATIONS TO COMPUTER SCIENCE

J. P. TREMBLAY
R. MANOHAR

McGRAW-HILL BOOK COMPANY
New York
St. Louis
San Francisco
Auckland
Düsseldorf
Johannesburg
Kuala Lumpur
London
Mexico
Montreal
New Delhi
Panama
Paris
São Paulo
Singapore
Sydney
Tokyo
Toronto

J. P. TREMBLAY
R. MANOHAR

Departments of Computational Science and Mathematics
University of Saskatchewan, Saskatoon

Discrete Mathematical Structures with Applications to Computer Science

This book was set in Modern 8A by Mono of Maryland.
The editors were Kenneth J. Bowman and Michael Gardner;
the production supervisor was Dennis J. Conroy.
The drawings were done by Eric G. Hieber Associates Inc.
Kingsport Press, Inc., was printer and binder.

Library of Congress Cataloging in Publication Data

Tremblay, Jean-Paul, date
Discrete mathematical structures with applications to computer science.

(McGraw-Hill computer science series)
Includes bibliographies.
1. Mathematics—1961- 2. Electronic data processing. 3. Machine theory. I. Manohar, R., date joint author. II. Title.
QA39.2.T72 510'.2'40901 74-23954
ISBN 0-07-065142-6

DISCRETE MATHEMATICAL STRUCTURES WITH APPLICATIONS TO COMPUTER SCIENCE

1 2 3 4 5 6 7 8 9 0 KPKP 7 9 8 7 6 5

PREFACE

Several advanced books in computer science begin with a chapter consisting of a selection of mathematical topics that the reader is assumed to know. The exposition of such topics is usually brief, and the principal results that are summarized become prerequisites for the remainder of the text. It is not possible to learn these topics from such a brief treatment. Nor is it possible for undergraduate students in computer science to study all the topics they are required to know by attending courses dealing with each individual topic as given by mathematics departments. In general, the trend is to select several topics in mathematics that are essential to the study of many computer science areas and to expose the students to the mathematical prerequisites in some other way. A similar development has occurred in most engineering curricula. In the same spirit, this book discusses certain selected topics in mathematics which can be referred to as "discrete mathematics." No prerequisites except the mathematical maturity of a high school student are assumed. Although many students taking a course in discrete mathematics may have had a freshman course in calculus, such a course is by no means a prerequisite to the study of this book. However, any additional mathematical courses taken by students will aid in their development of mathematical maturity.

It is not our intention to cover all topics in discrete mathematics. The omission of counting techniques, permutations, and probability will be felt by some readers. We have assumed that many high school students will have had some exposure to these topics.

The selection of the topics was governed by our desire to introduce most of the basic terminology used in as many advanced courses in computer science as possible. In order to motivate the students properly, we feel that it is important to consider certain applications as the terminology is introduced. There are several advantages in using this approach. Students begin to see the relevance of abstract ideas and are therefore better motivated. Moreover, they gain confidence in applying these ideas to solve practical problems.

We wish to emphasize that concepts and terminology should be introduced well before they are used. Otherwise, students must invariably struggle both with the basic tools and with the subject matter to which the tools are applied. Most of the material in this book is properly a prerequisite to so many computer science courses that it should be taught no later than at the sophomore level. The book has been written with this objective in mind.

The mathematical topics to be discussed are logic, set theory, algebraic structures, Boolean algebra, graph theory, and basic computability theory. Although well-known and excellent books exist in these areas, we introduce these topics still keeping in mind that the reader will eventually use them in certain practical applications particularly related to computer science. We have strived to introduce the theoretical material in a reasonably mathematically precise manner whenever possible, while avoiding long philosophical discussions, questions of paradoxes, and any axiomatic approach to certain theories. The topics selected will also support the more advanced courses in computer science programs such as in the areas of automata, computability, artificial intelligence, formal languages and syntactical analysis, information organization and retrieval, switching theory, computer representation of discrete structures, and programming languages. It is hoped that a grasp of the theoretical material in this book will permit a student to understand most of the mathematical preliminaries which are briefly discussed at the beginning of many articles and books in the areas of computer science just mentioned.

Because the relation between the mathematics and how or where it could be applied may not be clear to the reader, the computer representation of certain mathematical structures is discussed. The need for discrete structures in computer science is motivated by the selection of certain applications from various areas in the field. Algorithms are developed for most applications, and computer programs are given for some of them. The computer representation and manipulation of discrete structures such as strings, trees, groups, and plexes are not discussed in great detail, but only to the extent which permits the formulation of a solution to a particular application.

Chapter 1 discusses mathematical logic. An elementary introduction to certain topics in logic is given to students in education, commerce, economics, and social sciences in courses usually entitled "Finite Mathematics." However, such discussions usually end with the construction of truth tables, and in certain instances a brief introduction to the inference theory of the statement calculus is included. In order for students to be able to read technical articles and books in computer science, it is necessary for them to know something about predicate calculus. Therefore, we have gone further in our discussion of logic than is usually done in books on finite mathematics. Yet we have avoided the philosophical discussions and intricate details that are found in the books on mathematical logic meant for mathematicians and philosophers. The chapter contains a brief introduction to the application of logic to two-state devices.

Chapter 2 deals with set theory. Some mathematical rigor is maintained in the discussions and proofs are sometimes given, but we do not raise the question of paradoxes and the axiomatic approach to set theory. Sets, relations, orderings, and recursive functions are discussed. The computer representation and manipulation of certain structures are introduced in this chapter. An example of the interrelationship of set theory and logic is given. The topic of recursion (and its implementation) is dealt with in some detail since many programming languages permit its use. Furthermore, the concept of recursion is important in its own right because computer scientists will encounter, throughout their careers, problems where recursion is unavoidable. The chapter concludes with an algorithm for proving theorems in the propositional calculus.

Chapter 3 discusses algebraic structures. Most books in modern algebra devote almost all their attention to group theory while little is said about semigroups and monoids. The latter are also emphasized in this chapter since it is semigroup and monoid theory which is very important in certain areas of computer science such as formal language theory, syntactic analysis, and automata. This chapter contains a number of applications dealing with topics such as the compilation of Polish expressions, languages and grammars, the theory of fast-adders, and error detecting and correcting codes.

Chapter 4 is concerned with Boolean algebra and its application to switching theory and sequential machines. An introduction to the minimization of Boolean functions and to its use in the logical design of digital computer systems is given. Sequential machines and their equivalence are also discussed.

Chapter 5 gives a brief introduction to graph theory. Elements of graph theory are indispensable in almost all computer science areas. Examples are given of its use in such areas as syntactic analysis, fault detection and diagnosis in computers, and minimal-path problems. The computer representation and manipulation of graphs are also discussed so that certain important algorithms can be included.

Finally, Chapter 6 gives a very brief introduction to computability theory. The equivalence of finite-state acceptors and regular grammars is shown. Finally, the concept of an effective procedure is introduced. It is shown that a Turing machine can evaluate any partial recursive function.

The exercises are of both a theoretical and a programming nature and are meant to further the understanding of the application of the concepts to various areas of computer science. The material in this book incorporates, in addition to logic, most of what the ACM Curriculum Committee on Computer Science recommends for the course "Introduction to Discrete Structures."[1]

[1] Course B3 in CACM 11, pp. 172–173, 1968.

We hope that this book will be of use to computer scientists, engineers, nonmathematics students who desire an intermediate coverage of topics in discrete mathematics, and mathematicians who want to familiarize themselves with the application of the theory to computer science. Students who have some previous background in modern logic and algebra will be able to master the material in one semester. For other students who have no previous knowledge of logic and algebra, this book can be used in a two-semester course. Certain topics can be selected to form a one-semester course. The omission of the applications discussed in the text will not result in any loss of continuity in the material. This book is based on the experience gained in teaching a course on discrete structures at the University of Saskatchewan at Saskatoon during the past four years.

A basic familiarity with either standard FORTRAN or PL/I is assumed. PL/I is useful in applications that involve recursion or list structures.

We owe a great deal to John A. Copeck and Richard F. Deutscher, who made many valuable criticisms and suggestions throughout the entire preparation and proofreading of the book. They also helped us in formulating and testing most of the algorithms. In particular, John Copeck assisted in the preparation of Chapter 1 and Sections 2-7, 4-4, 4-5, 5-2, 5-4, and 6-2. Also, Richard Deutscher assisted in the preparation of Chapter 2, Sections 4-4 and 5-1, and many of the figures.

We also thank Peter Hardie for his assistance in working out the details on fault diagnosis in Section 5-4 and Andrew Carson for his suggestions in Chapter 3. Robert Probert proofread Sections 5-1 and 6-2, Don McKillican worked out of the exercises, and Gail Galgan helped in proofreading and constructing the index. We owe a very special thanks to Alice Mae MacDonald who did such an excellent job of typing the manuscript, and to Helen Henderson and Dorothy Peake for providing typing support. This work would not have been possible without the support given by the University of Saskatchewan.

J. P. TREMBLAY
R. MANOHAR

CONTENTS

Preface — **xiii**

1 **Mathematical Logic** — **1**

Introduction — **1**

1-1 Statements and Notation — 2

1-2 Connectives — 7

1-2.1 Negation — 8

1-2.2 Conjunction — 9

1-2.3 Disjunction — 10

1-2.4 Statement Formulas and Truth Tables — 11

Exercises *1-2.4* — 14

1-2.5 Logical Capabilities of Programming Languages — 14

1-2.6 Conditional and Biconditional — 18

Exercises *1-2.6* — 22

1-2.7 Well-formed Formulas — 23

1-2.8 Tautologies — 24

Exercises *1-2.8* — 26

1-2.9 Equivalence of Formulas — 26

1-2.10 Duality Law — 30

1-2.11 Tautological Implications — 32

	Exercises *1-2.11*	34
	1-2.12 Formulas with Distinct Truth Tables	35
	1-2.13 Functionally Complete Sets of Connectives	37
	Exercises *1-2.13*	38
	1-2.14 Other Connectives	39
	Exercises *1-2.14*	41
	1-2.15 Two-state Devices and Statement Logic	41
	Exercises *1-2.15*	49
	Exercises *1-2*	49
1-3	Normal Forms	50
	1-3.1 Disjunctive Normal Forms	50
	1-3.2 Conjunctive Normal Forms	52
	1-3.3 Principal Disjunctive Normal Forms	53
	1-3.4 Principal Conjunctive Normal Forms	56
	1-3.5 Ordering and Uniqueness of Normal Forms	58
	Exercises *1-3.5*	60
	1-3.6 Completely Parenthesized Infix Notation and Polish Notation	61
	Exercises *1-3.6*	64
1-4	The Theory of Inference for the Statement Calculus	65
	1-4.1 Validity Using Truth Tables	66
	Exercises *1-4.1*	67
	1-4.2 Rules of Inference	68
	1-4.3 Consistency of Premises and Indirect Method of Proof	72
	1-4.4 Automatic Theorem Proving	74
	Exercises *1-4*	79
1-5	The Predicate Calculus	79
	1-5.1 Predicates	80
	1-5.2 The Statement Function, Variables, and Quantifiers	82
	1-5.3 Predicate Formulas	85
	1-5.4 Free and Bound Variables	86
	1-5.5 The Universe of Discourse	88
	Exercises *1-5*	89
1-6	Inference Theory of the Predicate Calculus	90
	1-6.1 Valid Formulas and Equivalences	90
	1.6.2 Some Valid Formulas over Finite Universes	92
	1-6.3 Special Valid Formulas Involving Quantifiers	94
	1-6.4 Theory of Inference for the Predicate Calculus	96
	1-6.5 Formulas Involving More Than One Quantifier	99
	Exercises *1-6*	101
	Bibliography	102
2	**Set Theory**	**104**
	Introduction	**104**
2-1	Basic Concepts of Set Theory	105
	2-1.1 Notation	105

2-1.2 Inclusion and Equality of Sets 107
2-1.3 The Power Set 109
Exercises *2-1.3* 111
2-1.4 Some Operations on Sets 111
Exercises *2-1.4* 115
2-1.5 Venn Diagrams 116
Exercises *2-1.5* 118
2-1.6 Some Basic Set Identities 118
2-1.7 The Principle of Specification 121
2-1.8 Ordered Pairs and n-tuples 122
2-1.9 Cartesian Products 123
Exercises *2-1* 125
2-2 Representation of Discrete Structures 126
2-2.1 Data Structures 126
2-2.2 Storage Structures 129
2-2.3 Sequential Allocation 130
2-2.4 Pointers and Linked Allocation 132
2-2.5 An Application of Bit Represented Sets 141
Exercises *2-2* 147
2-3 Relations and Ordering 148
2-3.1 Relations 149
Exercises *2-3.1* 153
2-3.2 Properties of Binary Relations in a Set 154
Exercises *2-3.2* 155
2-3.3 Relation Matrix and the Graph of a Relation 156
2-3.4 Partition and Covering of a Set 162
Exercises *2-3.4* 164
2-3.5 Equivalence Relations 164
2-3.6 Compatibility Relations 171
Exercises *2-3.6* 175
2-3.7 Composition of Binary Relations 176
Exercises *2-3.7* 182
2-3.8 Partial Ordering 183
2-3.9 Partially Ordered Set: Representation and Associated Terminology 186
Exercises *2-3.9* 191
2-4 Functions 192
2-4.1 Definition and Introduction 192
Exercises *2-4.1* 197
2-4.2 Composition of Functions 198
2-4.3 Inverse Functions 201
Exercises *2-4.3* 204
2-4.4 Binary and n-ary Operations 205
Exercises *2-4.4* 210
2-4.5 Characteristic Function of a Set 210

	2-4.6 Hashing Functions	212
	Exercises *2-4.6*	219
	Exercises *2-4*	219
2-5	Natural Numbers	220
	2-5.1 Peano Axioms and Mathematical Induction	220
	2-5.2 Cardinality	224
	Exercises *2-5*	231
2-6	Recursion	232
	2-6.1 Recursive Functions, Sets, and Predicates	232
	Exercises *2-6.1*	243
	2-6.2 Recursion in Programming Languages	243
	Exercises *2-6*	259
2-7	Recursion in Mechanical Theorem Proving	261
	Exercises *2-7*	268
	Bibliography	268
3	**Algebraic Structures**	**270**
	Introduction	**270**
3-1	Algebraic Systems: Examples and General Properties	271
	3-1.1 Definition and Examples	271
	3-1.2 Some Simple Algebraic Systems and General Properties	274
	Exercises *3-1*	281
3-2	Semigroups and Monoids	282
	3-2.1 Definitions and Examples	282
	3-2.2 Homomorphism of Semigroups and Monoids	287
	3-2.3 Subsemigroups and Submonoids	292
	Exercises *3-2*	294
3-3	Grammars and Languages	294
	3-3.1 Discussion of Grammars	295
	3-3.2 Formal Definition of a Language	299
	3-3.3 Notions of Syntax Analysis	304
	Exercises *3-3*	308
3-4	Polish Expressions and Their Compilation	309
	3-4.1 Polish Notation	310
	3-4.2 Conversion of Infix Expressions to Polish Notation	311
	Exercises *3-4*	318
3-5	Groups	319
	3-5.1 Definitions and Examples	319
	Exercises *3-5.1*	328
	3-5.2 Subgroups and Homomorphisms	329
	Exercises *3-5.2*	333
	3-5.3 Cosets and Lagrange's Theorem	333
	3-5.4 Normal Subgroups	335
	3-5.5 Algebraic Systems with Two Binary Operations	341
	Exercises *3-5.5*	343

	Exercises *3-5*	344
3-6	The Application of Residue Arithmetic to Computers	344
	3-6.1 Introduction to Number Systems	345
	3-6.2 Residue Arithmetic	348
	Exercises *3-6*	359
3-7	Group Codes	359
	3-7.1 The Communication Model and Basic Notions of Error Correction	360
	3-7.2 Generation of Codes by Using Parity Checks	364
	3-7.3 Error Recovery in Group Codes	373
	Exercises *3-7*	376
	Bibliography	376
4	**Lattices and Boolean Algebra**	**378**
	Introduction	**378**
4-1	Lattices as Partially Ordered Sets	379
	4-1.1 Definition and Examples	379
	Exercises *4-1.1*	382
	4-1.2 Some Properties of Lattices	382
	Exercises *4-1.2*	385
	4-1.3 Lattices as Algebraic Systems	385
	4-1.4 Sublattices, Direct Product, and Homomorphism	387
	Exercises *4-1.4*	391
	4-1.5 Some Special Lattices	392
	Exercises *4-1.5*	397
4-2	Boolean Algebra	397
	4-2.1 Definition and Examples	398
	4-2.2 Subalgebra, Direct Product, and Homomorphism	401
	Exercises *4-2*	405
4-3	Boolean Functions	406
	4-3.1 Boolean Forms and Free Boolean Algebras	406
	4-3.2 Values of Boolean Expressions and Boolean Functions	410
	Exercises *4-3*	417
4-4	Representation and Minimization of Boolean Functions	418
	4-4.1 Representation of Boolean Functions	418
	4-4.2 Minimization of Boolean Functions	424
	Exercises *4-4*	433
4-5	Design Examples Using Boolean Algebra	436
	Exercises *4-5*	451
4-6	Finite-state Machines	453
	4-6.1 Introductory Sequential Circuits	453
	4-6.2 Equivalence of Finite-state Machines	456
	Exercises *4-6*	465
	Bibliography	466

5 Graph Theory 468

Introduction 468

5-1 Basic Concepts of Graph Theory 469

5-1.1 Basic Definitions 469

Exercises *5-1.1* 475

5-1.2 Paths, Reachability, and Connectedness 476

Exercises *5-1.2* 483

5-1.3 Matrix Representation of Graphs 484

Exercises *5-1.3* 493

5-1.4 Trees 494

Exercises *5-1.4* 501

5-2 Storage Representation and Manipulation of Graphs 501

5-2.1 Trees: Their Representation and Operations 501

5-2.2 List Structures and Graphs 509

Exercises *5-2* 514

5-3 Simple Precedence Grammars 515

5-3.1 Syntax Terminology 516

5-3.2 A View of Parsing 520

5-3.3 Notion and Use of Precedence Relations 523

5-3.4 Formal Definition of Precedence Relations 526

5-3.5 Parsing Algorithm for Simple Precedence Grammars 530

Exercises *5-3* 532

5-4 Fault Detection in Combinational Switching Circuits 533

5-4.1 Faults in Combinational Circuits 534

5-4.2 Notions of Fault Detection 535

5-4.3 Algorithm for Generating a Fault Matrix 537

5-4.4 Procedure for the Detection of Faults 547

Exercises *5-4* 549

5-5 PERT and Related Techniques 550

Exercises *5-5* 555

Bibliography 555

6 Introduction to Computability Theory 557

Introduction 557

6-1 Finite-state Acceptors and Regular Grammars 558

Exercises *6-1* 566

6-2 Turing Machines and Partial Recursive Functions 566

Exercises *6-2* 584

Bibliography 584

Appendix 585

Index 591

1
MATHEMATICAL LOGIC

INTRODUCTION

One of the main aims of logic is to provide rules by which one can determine whether any particular argument or reasoning is valid (correct).

Logic is concerned with all kinds of reasonings, whether they be legal arguments or mathematical proofs or conclusions in a scientific theory based upon a set of hypotheses. Because of the diversity of their application, these rules, called rules of inference, must be stated in general terms and must be independent of any particular argument or discipline involved. These rules should also be independent of any particular language used in the arguments. More precisely, in logic we are concerned with the forms of the arguments rather than with the arguments themselves. Like any other theory in science, the theory of inference is formulated in such a way that we should be able to decide about the validity of an argument by following the rules mechanically and independently of our own feelings about the argument. Of course, to proceed in this manner requires that the rules be stated unambiguously.

Any collection of rules or any theory needs a language in which these rules or theory can be stated. Natural languages are not always precise enough. They are also ambiguous and, as such, are not suitable for this purpose. It is therefore necessary first to develop a formal language called the *object language*. A formal language is one in which the syntax is well defined. In fact, every scientific discipline develops its own object language which consists of certain well-defined terms and well-specified uses of these terms. The only difference between logic and other disciplines is that in other disciplines we are concerned with the use of the object language while in logic we are as interested in analyzing our object language as we are in using it. In fact, in the first half of this chapter we shall be concerned with the development and analysis of an object language without considering its use in the theory of inference. This study has important applications in the design of computers and several other two-state devices, as is shown in Sec. 1-2.15. We emphasize this part of logic because the study of formal languages constitutes an important part in the development of means of communication with computing machines. This study is followed by the study of inference theory in Sec. 1-4. It soon becomes apparent that the object language developed thus far is very limited, and we cannot include some very simple argument forms in our inference theory. Therefore, in Sec. 1-5 we expand our object language to include predicates, and then in Sec. 1-6 we discuss the inference theory of predicate logic.

In order to avoid ambiguity, we use symbols which have been clearly defined in the object languages. An additional reason to use symbols is that they are easy to write and manipulate. Because of this use of symbols, the logic that we shall study is also called *symbolic logic*. Our study of the object language requires the use of another language. For this purpose we can choose any of the natural languages. In this case our choice is English, and so the statements about the object language will be made in English. This natural language (English) will then be called our *metalanguage*. Certain inherent difficulties in this procedure could be anticipated, because we wish to study a precise language while using another language which is not so precise.

1-1 STATEMENTS AND NOTATION

In this section we introduce certain basic units of our object language called primary (primitive, atomic) statements. We begin by assuming that the object language contains a set of declarative sentences which cannot be further broken down or analyzed into simpler sentences. These are the primary statements. Only those declarative sentences will be admitted in the object language which have one and only one of two possible values called "truth values." The two truth values are *true* and *false* and are denoted by the symbols T and F respectively. Occasionally they are also denoted by the symbols 1 and 0. The truth values have nothing to do with our feelings of the truth or falsity of these admissible sentences because these feelings are subjective and depend upon context. For our purpose, it is enough to assume that it is *possible* to assign one and only one of the two possible values to a declarative sentence. We are concerned in our study with the *effect* of assigning any particular truth value to declarative sentences rather than with the *actual* truth value of these sentences. Since we admit only two possible truth values, our logic is sometimes called a *two-valued logic*.

We develop a mechanism by which we can construct in our object language other declarative sentences having one of the two possible truth values. Note that we do not admit any other types of sentence, such as exclamatory, interrogative, etc., in the object language.

Declarative sentences in the object language are of two types. The first type includes those sentences which are considered to be primitive in the object language. These will be denoted by distinct symbols selected from the capital letters $A, B, C, \ldots, P, Q, \ldots$, while declarative sentences of the second type are obtained from the primitive ones by using certain symbols, called connectives, and certain punctuation marks, such as parentheses, to join primitive sentences. In any case, all the declarative sentences to which it is possible to assign one and only one of the ~~two~~ possible truth values are called *statements*. These statements which do not contain any of the connectives are called *atomic* (*primary*, *primitive*) *statements*.

We shall now give examples of sentences and show why some of them are not admissible in the object language and, hence, will not be symbolized.

1 Canada is a country.
2 Moscow is the capital of Spain.
3 This statement is false.
4 $1 + 101 = 110$.
5 Close the door.
6 Toronto is an old city.
7 Man will reach Mars by 1980.

Obviously Statements (1) and (2) have truth values *true* and *false* respectively. Statement (3) is not a statement according to our definition, because we cannot properly assign to it a definite truth value. If we assign the value *true*, then Sentence (3) says that Statement (3) is false. On the other hand, if we assign it the value *false*, then Sentence (3) implies that Statement (3) is true. This example illustrates a semantic paradox. In (4) we have a statement whose truth value depends upon the context; viz., if we are talking about numbers in the decimal system, then it is a false statement. On the other hand, for numbers in binary, it is a true statement. The truth value of a statement often depends upon its context, which is generally unstated but nonetheless understood. We shall soon see that we are not going to be preoccupied with the actual truth value of a statement. We shall be interested only in the fact that it *has* a truth value. In this sense (4), (6), and (7) are all statements. Note that Statement (6) is considered true in some parts of the world and false in certain other parts. The truth value of (7) could be determined only in the year 1980, or earlier if a man reaches Mars before that date. But this aspect is not of interest to us. Note that (5) is not a statement; it is a command.

Once we know those atomic statements which are admissible in the object language, we can use symbols to denote them. Methods of constructing and analyzing statements constructed from one or more atomic statements are discussed in Sec. 1-2, while the method of symbolizing atomic statements will be described here after we discuss some conventions regarding the use and mention of names in statements.

It is customary to use the *name* of an object, not the object itself, when making a statement about the object. As an example, consider the statement

8 This table is big.

The expression "this table" is used as a name of the object. The actual object, namely a particular table, is not used in the statement. It would be inconvenient to put the actual table in place of the expression "this table." Even for the case of small objects, where it may be possible to insert the actual object in place of its name, this practice would not permit us to make two simultaneous statements about the same object without using its name at one place or the other. For this reason it may be agreed that a statement about an object would contain never the object itself but only its name. In fact, we are so familiar with this convention that we take it for granted.

Consider, now, a situation in which we wish to discuss something about a *name*, so that the name is the object about which a statement is to be made. According to the rule just stated, we should use not the name itself in the statement but some name of that name. How does one give a name to a name? A usual method is to enclose the name in quotation marks and to treat it as a name for the name. For example, let us look at the following two statements.

9 Clara is smart.
10 "Clara" contains five letters.

In (9) something is said about a person whose name is Clara. But Statement (10) is not about a person but about a name. Thus "Clara" is used as a name of this name. By enclosing the name of a person in quotation marks it is made clear that the statement made in (10) is about a name and not about a person.

This convention can be explained alternatively by saying that we *use* a certain word in a sentence when that word serves as the name of an object under consideration. On the other hand, we *mention* a word in a sentence when that word is acting not as the name of an object but as the name of the word itself. To "mention" a word means that the word itself has been converted into an object of our consideration.

Throughout the text we shall be making statements not only about what we normally consider objects but also about other statements. Thus it would be necessary to name the statements under consideration. The same device used for naming names could also be used for naming statements. A statement enclosed in quotation marks will be used as the *name* of the statement. More generally, any expression enclosed in quotation marks will be used as the name of that expression. In other words, any expression that is mentioned is placed in quotation marks. The following statement illustrates the above discussion.

11 "Clara is smart" contains "Clara."

Statement (11) is a statement about Statement (9) and the word "Clara." Here Statement (9) was named first by enclosing it in quotation marks and then by using this name in (11) along with the name "Clara"!

In this discussion we have used certain other devices to name statements. One such device is to display a statement on a line separated from the main text. This method of display is assumed to have the same effect as that obtained