TEACHING COMPUTER PROGRAMMING IN INTERMEDIATE SCHOOLS

A thesis presented in partial
fulfilment of the requirements
for the degree of Doctor of Philosophy
in Education
at Massey University

Richard John Spence

1975

Abstract

This investigation concerned classroom learning of a computer programming course by Form 2 pupils in New Zealand Intermediate schools. Samples were employed representing the full range of ability levels found in such schools.

The programming task was divided into a pre-coding phase and a coding phase, and the capacity to perform the tasks relating to each of the two phases were postulated as separate abilities. This division was shown to be justified. Nevertheless, measures of the two abilities were found to be moderately correlated, and each also correlated moderately with a measure of mathematical attainment. Analysis of the results showed that these correlations were not due to general intelligence alone. The fine structure underlying the relationships was also examined.

In the study, it was further shown that three measures of academic achievement predicted attainment in the programming course more effectively than fourteen personality measures. Some similarities and some differences were discovered between the results of this prediction study and similar studies with adults.

Finally, two different teaching sequences were compared against each other and with a control group. It was established that mastery of the pre-coding phase of programming was improved by teaching, but that the place in the course where this teaching was given made no significant difference to end-of-course achievement. On the other hand, altering the timing of the instruction in the elements of the programming language was found to produce a significant difference in mastery of coding skills.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acknowledgements

The writer thanks his supervisors, Professor C.G.N. Hill, Dr D.M. McAlpine and Dr R.E. Sass of Massey University's Department of Education for their help. Their valuable suggestions and patient reading of earlier drafts of this thesis are especially appreciated.

Thanks are also due to Mr J. Dowling, formerly District Senior Inspector of Schools for encouraging the project, and to the Principals of the three participating schools: Messrs R. Ward, J. Morris and P. McCarthy.

To those others in industrial, academic and educational circles who have made varied contributions, the writer acknowledges profound indebtedness.

PREFACE

The burgeoning growth of computer studies in primary and secondary schools makes research into its educational significance a matter of urgency. The growth has been accelerated by the falling cost of computer hardware and the increasing number of teachers, particularly of mathematics, who have some computing in their background and a desire to teach it. Since the United States is the world's major manufacturer of computing equipment, it is there that developments have been most spectacular: the Survey of Computing Activities in Secondary Schools (Darby et al., 1970) gives a detailed account of the extent and character of the growth in that country. But the growth has also been described by the OECD Centre for Educational Research and Innovation as occurring

> "... on a haphazard and random basis. In the main, the impetus has come from individuals, many of whom were interested in only one facet, for example, the use of computers in the teaching of mathematics. One result of this is that in too many schools, computer education is restricted to mathematics lessons or even to mathematically-gifted pupils. When this happens, many of the most valuable rewards of computer education are lost." (OECD, 1971)

Well-founded research will enable educators to decide whether or not computer education in schools really would produce worthwhile outcomes. Certainly, programming enjoys widespread support as a subject for study in schools: The quotations that follow are typical of almost daily utterances favouring the introduction of programming into the school curriculum:

> "In the preparation of a program, logical thought, care, and precision are required ... The discipline of systematic thinking and clear communication that are associated with the logical aspect of computer work are themselves of educational value." (Scottish Ed. Dept., undated)

> "The disciplines of problem definition, systems analysis and design, flowcharting and programming have been shown to significantly develop the child's ability to approach situations in such a confident, ordered, and creative way."
> (ICL/CES, 1972)

> "... I propose creating an environment in which the child will become highly involved in experiences of a kind to provide rich soil for the growth of institutions and concepts for dealing with thinking, playing and so on (through computer programming) ... the empirical evidence is very strong that we can do it ..."
> (Papert , 1971)

Statements like the above share a common characteristic: belief in an undefined "thinking" skill and an unspecified mechanism by which computer programming activities stimulate it. Because the advocates of programming as a school subject have so far concerned themselves with demonstrating that programming <u>can</u> be taught at quite junior grade levels, little has been done to investigate the proper place of programming in the school curriculum. At the same time, little is known empirically about how best to teach programming, and to whom. Cox (1972) has written:

"In a recent search of the literature it was surprising and disconcerting to find little on the effectiveness of teaching programming. Most studies concentrated on the tools used rather than on the problem, though Sackman et al. (1968) document what any manager knows: the gulf between the good and the mediocre programmer .... it is high time we found answers to some of the questions by controlled experiments with a wide variety of groups."

A main aim of the present study was to provide some of those answers.

Explanation of Technical Terms

Studies of computer education cross the boundaries of Education and Computer Science, and therefore draw on the terminologies of both. Because computer terms will be used throughout this account, and because they are essential to the discourse, the following explanation is provided at this point rather than in an appendix:

An ELECTRONIC DIGITAL COMPUTER is an assembly of interconnected devices, mainly electronic but also electromechanical, capable of performing a wide variety of functions on information that is presented in the form of CHARACTERS (letters, digits, punctuation marks). The business of getting information into the computer is called INPUT, and the same word is used without ambiguity to denote the ingoing information itself. (The term DATA is strictly equivalent to "information", but is most usually encountered in relation to input.) Once information has been input, the computer can PROCESS it and ultimately OUTPUT the modified information. The processing phase includes all the data-manipulation (e.g. arithmetic), logical tests, and management of the computer's memory, or STORE, where information is held while it is being worked on. By means of INSTRUCTIONS to the computer, a human can control the input, processing and output that occurs.

A set of instructions to perform a specific function is called a PROGRAM. However, as it can modify the data in its memory, the computer can also modify its program; it is because of this distinctive feature that modern digital computers are sometimes called STORED-PROGRAM computers. Because the program is stored, its sequence can be altered by special BRANCH instructions incorporated in the program itself. An important use of branch instructions is to create LOOPS in programs whereby certain sequences of instructions are operated repeatedly. Programs will also be self-modifying if they contain CONDITIONAL INSTRUCTIONS

which the computer only obeys under certain specified conditions. All non-trivial programs use conditional, branch, and conditional branch instructions.

PROGRAMMING[1] involves converting a problem into a set of directions to a computer to solve it. The function is sometimes broken down into several parts, particularly if the problem is very complex. A specific procedure for solving a problem is an ALGORITHM[1]. The process of writing the detailed step-by-step instructions for the computer to follow is CODING[1]. After a program is written, it is tested by letting it perform its function on test data to which the proper solution is known. This process is PROGRAM CHECKOUT or DEBUGGING; when it is done using pencil and paper rather than the computer itself, it is called a DESK-CHECK.

Careful and complete algorithm design and desk-checking must be carried out in advance if a program is to be run on a BATCH-PROCESSING system - one in which completed programs and data must be batched together and input to the computer at one time. Batch-processing, especially when one batch contains numerous programs, is cheap and efficient in computer terms but necessarily involves some delay between preparation of a program and receipt of the resulting output. The delay may sometimes be as little as an hour (e.g. for a high-priority user in a university computer unit), but is commonly a day or more. In contrast to batch-processing, CONVERSATIONAL PROGRAMMING gives the programmer direct access to the "live" computer system via an interactive terminal, enabling him to get instant results even from incomplete programs.

The programmer will be expected to produce some descriptions of his program and how it operates so that others may understand how it works. This DOCUMENTATION may include a FLOWCHART[1]: a graphic description or diagram of the various paths and branches followed by the program.

---

[1] This important term is defined in detail in Chapter II

The repertory of instructions available to the programmer for a specific computer is that computer's MACHINE LANGUAGE. (Because this use of the word "language" is somewhat misleading, human languages such as English are distinguished as NATURAL LANGUAGES.) HIGHER-LEVEL LANGUAGES have been developed to help the programmer by simplifying the tedious aspects of writing machine language; these include FORTRAN and ALGOL for scientific work, COBOL for business data processing, and dozens of others each with its own special uses and characteristics. Generally speaking, the higher-level languages can be used on a wide range of models and makes of computers, provided appropriate COMPILERS are available. A compiler is a master-program that converts a higher-level language into machine language; programs that perform similar functions at a much simpler level are ASSEMBLERS.

SOFTWARE is the term used to denote the totality of programs, documentation and procedures required in order to use a computer; sometimes it is used more specifically to mean those programs of general usefulness (such as compilers) that are available to all users. Software is contrasted to HARDWARE - the physical machinery itself, which is controlled not by the programmer but by a COMPUTER OPERATOR, whose job includes such matters as inputting the computer's work and collecting the output to be returned to the user. Though in real-life computing it is unusual for an operator to engage in programming, he does make use of a master item of software called the OPERATING SYSTEM that helps him in sequencing jobs, accounting, and calling up other software.

CHAPTER I

INTRODUCTION

This study was intended to examine the justification for applying some of the more prevalent items of computer folk-wisdom to computer education in schools. The beliefs in question, which enjoy widespread currency, were

> Being a good coder does not imply being good at developing algorithms, and vice versa.
>
> Attainment in computer programming is unrelated to attainment in mathematics.
>
> Personality is just as important in predicting success at programming as is past academic attainment.
>
> Programming is learnt, not taught; therefore the strategy an instructor uses in presenting his material makes no difference to the student's attainment at the end of the course.

In ten years spent in the company of computer professionals, the writer heard all these beliefs expounded many times over. However, enough objections had been raised to each one of these 'rules' to show that they were not universally accepted in the industry, even for professional programming which is widely practised and at least fairly well documented. And even if they did apply for computer professionals, there could be no justification for applying them without validation to the school situation. Validation was the major aim of this study.

The need for such a study stemmed also from the general paucity of information about <u>any</u> aspects of computer programming as a school subject in its own right. The researcher was motivated to undertake the research when, after seven years of teaching programming at Form 2, 3, 6, 7 and University Extension levels, he was unable to discover either from published research or his

own experience whether the four folk-myths above, whether valid or otherwise for adult programmers might be valid for his younger students.

Children at the Form 2 level in New Zealand intermediate schools particularly interested the researcher, as this group appears to be the youngest to whom programming can profitably be taught in a batch-processing environment. There is some informal evidence of differences between the performance of this age-group and older students: an apparently greater facility in learning the basics of low-level coding, for example (Spence, 1974). Hatfield and Kieren (1972) have conjectured that the transition, at around 12-13 years, between Piaget's stages of concrete and formal operations may be involved.

The present study, by concentrating on what seems to be the low end of the age spectrum, should yield results tending either to confirm, or to contrast with, the existing body of knowledge about undergraduate and adult programming. Such information would not only lead to useful insights into the general psychology of computer programming, but also provide a basis for planning computer science curricula through the secondary school.

In general terms, the aim of this study was to provide needed research in the field of school computer studies. Important problems have been posed by Cox (1972):

> "Of course there are several problems to overcome, the most severe of which is the length of time it takes to correct and assess programs."

by Hatfield and Kieren (1972):

> "What is the effect of programming on the development of certain mathematical skills?"

by C.A. Alspaugh (1972):

> "... due to the particular personality factors that appear to influence achievement in computer programming, it would not be realistic to assume that all students who are talented in mathematics will also be talented in computer programming."

and by J.W. Alspaugh (1971):

> "... further investigation should include a number of variations in the instructional schemes used ... In addition, sample sizes should be increased to enable one to investigate treatment by ability level interactions ...".

The programme of research undertaken in this investigation was directed towards answering the questions expressed or implied in the above quotations.

Four specific goals were identified:

1. to construct objective pencil-and-paper tests measuring the important outcomes sought for programming courses;
2. to examine the relationships between achievement in a programming course and concurrent achievement in mathematics generally, and in the specific aspects of Numerical Relationships, Spatial Relationships, Logical Relationships and Mechanical Arithmetic;
3. to predict achievement in a programming course from past achievement in school subjects other than programming, and from scores on a personality test;
4. to compare the effects on end-of-course achievement on a programming course of teaching

   (a) flowcharting before coding, and

   (b) coding before flowcharting,

   also to comparte the effects obtained by each of these strategies with those of not teaching programming at all.

In undertaking such a programme, much work of a pioneering nature was required. Objective measures of programming achievement had to be developed by the researcher - few existed even for adults, none at all for twelve-year-olds. Therefore a major effort was needed in designing, testing, revising and finally standardizing new tests for use in the study.

Before the relationship of programming to mathematics could be investigated, a suitable test of mathematical attainment had also to be constructed, though this task was much simpler as adequate precedents existed.

Finally, the absence of a corps of trained teachers of programming added another constraint. To deal with this, the researcher undertook to teach all the programming courses used in the investigation. In this way, the uniformity of course presentation was assured, as was adherence to the carefully-designed plans of instruction used in the comparative study.

The general plan of the research programme is summarized in Figure I.

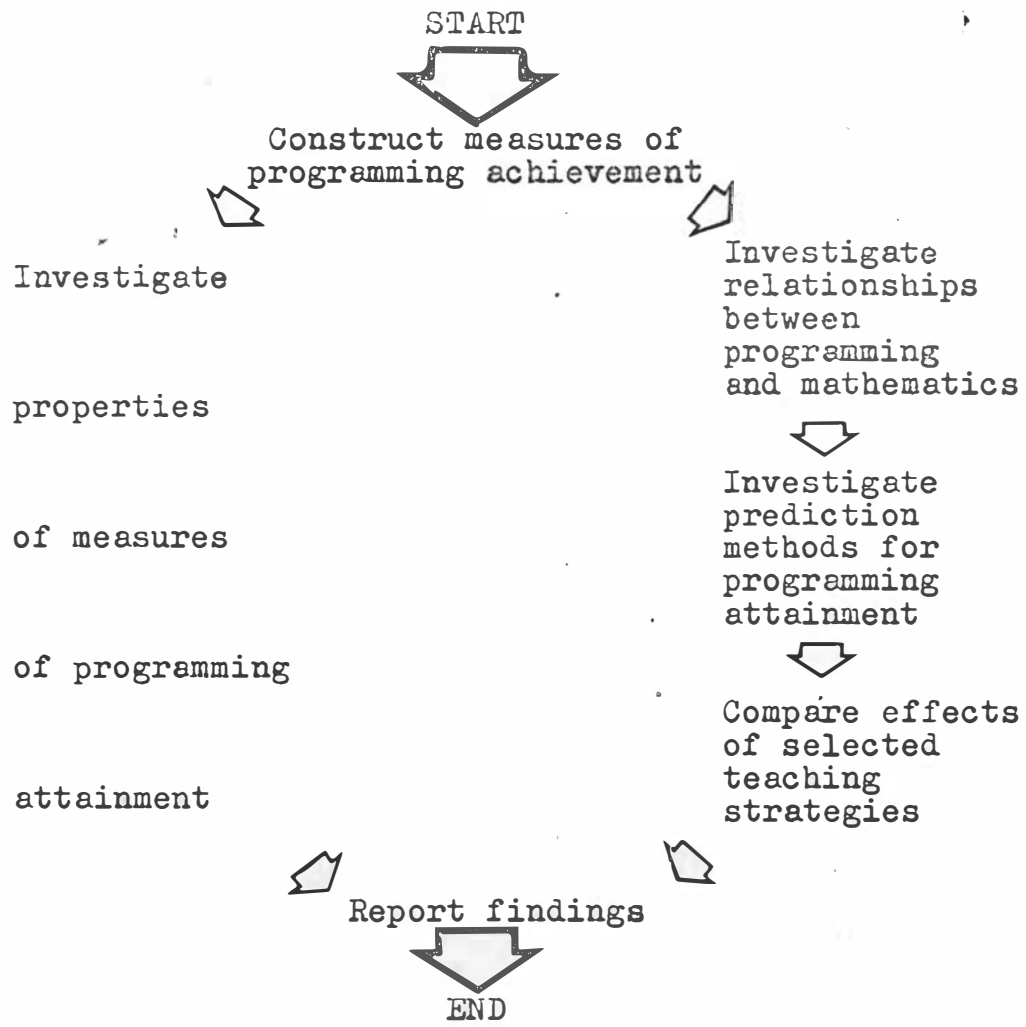


Figure I: Diagram showing General Plan of Research Programme.

Review of the Literature

Since computer programming is so new in the schools, there is a serious shortage of research literature directly related to the present investigation.

On the prevalence of computer courses in schools, the most serious study for a major country is the Survey of Computing Activities in Secondary Schools (Darby et al., 1970), which found such courses in 1599 U.S. schools. For New Zealand, the number of schools offering computing studies probably exceeds 200 (Spence, 1975).

Apologists for programming as a contributing activity for mathematics learning have pointed to specific areas of use in that subject. For example:

> "The elusive notion of variable becomes more operant as the student deals with a system in which a symbol (variable) denotes any member of a set of replacements ..." (Hatfield and Kieren, 1971)
>
> "Students will prform the calculations of a program in order to check the validity of the algorithm." (Hatfield, 1966)
>
> " ... we must think in terms of operators rather than operations in arithmetic, and we can consider algebraic expressions as the language of communication." (Smith and Smith, 1971)

How contributions like these can be exploited through extended mathematics courses is shown in detail in such texts as the Computer Assisted Mathematics Program series (Johnson et al., 1968 -) which relies on a real computer, and Smith + Smith's Computer Age Mathematics (1971) which uses an imaginary one.

Studies by Bitter (1970), Fielder (1969), Haven (1968, 1970), Holoien (1970), Wallace (1968), Hatfield and Kierenn (1972) have confirmed empirically that integrating programming into school mathematics programmes can significantly improve students' mathematical achievement

and hence is a useful device for teaching that subject. (It may be equally useful for teaching English composition or modern languages.) They did not examine programming as a school subject in its own right, nor did they reveal whether the relationship between programming and mathematics is real or illusory, whether it is special or part of some more general relationship.

Evidence of an implicit belief that computer studies relate closely and fairly exclusively to mathematics was given in the Survey of Computing Activities in Secondary Schools. In U.K. schools,

> "Applications of computers to mathematics instruction dominated. Almost three-quarters of all computer applications were involved with mathematics instruction." (Darby et al., 1970)

New Zealand secondary school pupils unexposed to programming courses also believed that computing was closely related to mathematics, according to a local survey.(Spence, 1970). In 1972, when the present study commenced, there was still a notable lack of evidence confirming or denying that the relationship between computing and mathematics was simply due to

> " ... a historical accident that, in the 1940's, the problems with which it was felt that electronic computers could particularly assist were largely numerical in character ..." (Martin, 1972)

Before a rationale for programming in schools could be arrived at, objective evidence on these issues was required. One aim of this study was to provide tools for obtaining some of that evidence.

The main new tools required were instruments to measure programming proficiency. G.M. Weinberg (1971) has discussed this problem in his penetrating work, The Psychology of Computer Programming :

> "The question of what makes a good program is not a simple one, and may not even be a proper question.

> Each program has to be considered on its own merits and in relation to its own surroundings. Some of the important factors are:
>
> 1. Does the program meet specifications? Or rather, how well does it meet specifications?
> 2. Is it produced on schedule, and what is the variability in the schedule that we can expect from particular approaches?
> 3. Will it be possible to change the program when conditions change? How much will it cost to make the change?
> 4. How efficient is the program, and what do we mean by efficiency? Are we trading efficiency in one area for inefficiency in another?
>
> In the future, and particularly in the discussion of this book, we should refrain from using the concept 'good program' or 'good programmer' as if it were something universally agreed upon, or something that even <u>can</u> be universally agreed upon, or something that even <u>should</u> be universally agreed upon." (p.25)

While some of Weinberg's criteria do not apply in the school situation (and Weinberg has been careful to distinguish between the working professional programmer, the trainee professional programmer and the programming amateur), his main point was considered to apply: that opinions on what is a good program or a good programmer are likely to vary very widely from instructor to instructor or from supervisor to supervisor, depending on which aspects of programming they choose to emphasize. This fact limits the generality of any study that uses a single score as a measure of programmer efficiency or of achievement in a programming course.

In a later chapter, Weinberg has proposed subdividing the programming task into stages, each stage requiring a different sort of skill:

> "The job of system design calls for an eye which never loses sight of the forest, whereas the job of debugging may require that every tree - even every branch or leaf - be seen with utmost clarity. The job of coding often requires squeezing out every drop of redundancy, and the job of doumentation may require that simple sentences be plumped up to paragraph size." (1971, p.132)

He has used this view as a rationale for the team approach to professional programming, in which different programmers concentrate on different stages of the programming task. The same writer has also pointed out that different programming projects, especially small ones, tend to be dominated by performance on different phases of the work, and that this effect alone could account for very great differences in measured performance.

Concerned with grading programs rather than programmers, Clutterham (1970) has developed a tool which is important because of its objectivity and convenience. It is a compiler for student programs written in a language similar to CESIL (see Chapter III below); during compilation and execution it also grades each program automatically. The raw grade produced is an arbitrary function (but one possessing considerable face validity) of

1. the extent to which the program meets the instructor's specifications of the job it is supposed to do;
2. the manner in which the execution ended (i.e. whether through normal exit or because of an invalid instruction, excessive runtime, etc.);
3. the time it took to execute the program; and
4. the number of instructions in the program.

The function measuring and combining these elements was to be determined in form by Clutterham, and in its parameters by the individual instructor, the parameters being set afresh for each new task.

The fact that Clutterham's scoring function was built up from the four variables listed above showed that like Weinberg he believed a multiplicity of factors to be involved in judging the quality of a program. Again, by providing for variation of the parameters in the scoring function, Clutterham has supported Weinberg's view that judgements about the quality of a program depend on the environment in which it is produced and the pre-occupations of the instructor. Where Weinberg and Clutterham seem to have parted company is in Clutterham's production of a single number as the final score for a program; in this respect, like all the other writers here reviewed, Clutterham has shown himself to be at variance with Weinberg's fundamental thesis that the components of the programming act should be viewed separately rather than jointly. The present study used a two-dimensional view of programming, and followed Weinberg's scheme in dividing the act into a pre-coding phase and a coding phase.

A search of the literature for experimental evidence of the effectiveness of strategies for programming instruction, proved almost as barren as Cox's (above, p.ix). The relationship of grade placement to programming aptitude and Fortran programming achievement had been investigated by J.W. Alspaugh (1971), who concluded that his results

> "(a) indicate that a student's programming aptitude score (on the APPT) increases with grade level, and
>
> (b) imply that the grade placement of beginning Fortran courses can be lowered from grades 15 - 16 to grades 11 - 12 with comparable achievement if the instruction time is increased."

The value of Alspaugh's study was limited because it concentrated on academically talented, volunteer students, and because he changed two of his variables (grade and teaching method) simultaneously. Whether the younger

students would have performed differently under conditions identical to those experienced by the older students was not investigated.

Unlike teaching, the business of predicting programmer performance is economically profitable and therefore well documented so far as adults and college students are concerned. The ATPP manual (IBM, 1964b), the CPAB manual (Palermo, 1967a), Seiler (1965) have described prediction studies using professional or apprentice programmers as subjects and supervisor ratings of job performance as criteria, reporting predictor/criterion correlations ranging from 0.29 to 0.62. As Weinberg (1971) has pointed out, performance in programming courses should be more predictable than job performance, as the following studies have confirmed:

Table 1: Summary Details of Twelve Prediction Studies of Programming Performance, describing Sample and Criterion Measure Used, and Correlation Coefficients obtained between Predictor and Criterion Measure.

Study	Sample	Criterion	Predictor/Criterion Correlation
ATPP Standardisation (IBM) 1964)	Programmer trainees	Training course grades	0.45 - 0.64
CPAB Standardisation (Palermo, 1967a)	Programmer trainees	Training course grades	0.52 - 0.71
Wolfe (1974)	College men	College programming course grades	0.45
———	College women	College programming course grades	0.70
Katz (1962)	Army EDP trainees	Training course grades	0.68
Stalnaker (1965)	College Students	College programming course grades	0.81
Bauer et al. (1968)	College Students	College programming course grades	0.81
C.A. Alspaugh (1972)	College Students	College programming course grades	0.58 - 0.63
Wolfe (1971)	High School Students	School programming course grades	0.68
———	Private EDP school students	EDP school programming course grades	0.41
———	Junior College students	College programming course grades	0.54
Howell et al. (1967)	Programmer trainees	Robot Test	0.60 - 0.71

Rowan (I957) experimented with 30 tests of various aptitudes and traits. He rejected 28 of them as predictors of programming achievement, leaving the Thurstone Primary Abilities Test and the Thurstone Temperament Schadule. A multiple regression analysis " .. resulted in a correlation of 0.54 between criterion (supervisor ratings) and subtests labelled 'Verbal Meaning', 'Reasoning' and 'Emotional Stability' ... The multiple correlation remained 0.54 when one other test, involving the perception of spatial relations, was included."

Subsequent studies by Bauer et al. (I968), C.A. Alspaugh (I972), Howell et al. (I967) and Morris and Parkinson (I97I), have emphasised past academic success as a predictor, and more specifically mathematical background. Within the mathematical field, interest in, and skill at, numerical reasoning were found to be significant predictors in all four studies. Bauer et al. (I968) and Howell et al. (I967) also found the ATPP Figure Series test of spatial reasoning a significant predictor. Verbal skills were mentioned by Rowan alone.

The sub-sections of three published and two widely-used but unpublished predictive tests[I] of programmer aptitude were categorised as "directly predictive" and indirectly predictive". Direct predictors were those considered to sample behaviour closely resembling the programming activity itself; others were classified indirect.

I These unpublished tests are the property of two major computer manufacturers (not IBM) who prefer to be unnamed. The tests are known in their respective companies as the E-5I and the CPST

Table 2: Content of Five Programmer Aptitude Tests, by Test Title and Type of Item.

	ATPP (IBM 1964a)	CPAB (Palermo, 1964)	Wolfe (1972)	CPST	E-51
DIRECT PREDICTORS					
Translating word descriptions into special (eg Mathematical notation		Part 2 ('reasoning')		Part 2 ('reasoning')	
Predicting the outcome of a program written in computer-like code			whole test		
Flowcharting		Part 5 ('diagramming')		Part 5 ('diagramming')	whole test
INDIRECT PREDICTORS					
Numerical Reasoning	Part 3 ('arithmetical reasoning')	Part 4 ('number ability')		Part 4 ('number ability')	
Letter Series	Part 1 ('letter series')	Part 3 ('letter series')		Part 3 ('letter series')	
Vocabulary		Part 1 ('verbal meaning)		Part 1 ('verbal meaning')	
Spatial Reasoning	Part 2 ('figure series')				

Morris and Wise (I972), Morris and Parkinson (I97I) and C.A. Alspaugh (I972), who have studied the personality characteristics relating to success in programming, use the following descriptions for traits directly related to such success:

Social presence: poised and selfconfident	(Morris and Wise)
Reflective	(Alspaugh)
Achievement via Independence	(Morris and Wise)
Flexibility	(Morris and Wise)

while they report the following as being inversely related to programming success:

Impulsive	(Alspaugh)
Responsibility: conformity, conscientiousness, dependability, alert to moral and ethical values	(Morris and Wise)
Persuasive	(Morris and Parkinson)
Sociable	(Alspaugh)

CHAPTER II

DEFINITIONS, ASSUMPTIONS AND HYPOTHESES

Definitions of Programming Terms

The term "programming" means different things to different people. To some, it means just coding, while to others like the Software Writing Group of Digital Equipment Corp., (1970), it means as wide a range of activities as:

"1. Definition of the problem to be solved,

2. Determination of the most suitable solution method,

3. Design and analysis of the solution - flowcharting,

4. Coding the solution in the programming language, and

5. Program checkout. " (pp. 3-2, 3-3)

In this investigation a middle line was taken, bearing in mind that the programming under consideration is that which takes place in a schoolroom. In such circumstances, problem definition is part of the teacher's role, and systematic program checkout is seldom taught or practised. Therefore points 1 and 5 were inappropriate for the present study, and the following definition was adopted:

Programming: That activity occurring after an explicit statement of the problem has been obtained either verbally or in equations, and including the determination of the most suitable solution method, flowcharting, and coding.

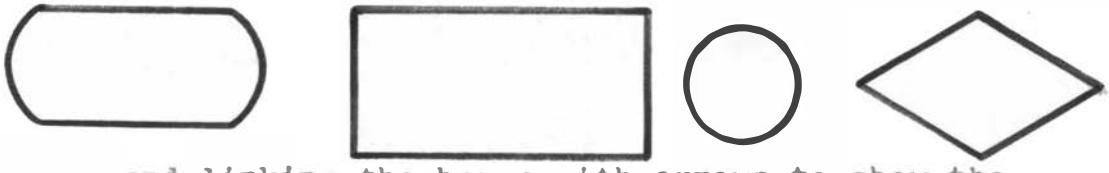
(Note: This definition is almost identical to one used in a 1956 study by the RAND Corp. Rowan (1957))

Although opinion varies on what the word "programming" means, it is believed that the following functional definitions of individual aspects of programming would meet with general approval:

Algorithm: Unambiguous communication of a finite set of actions required to perform a desired task in a finite number of steps, the set of actions to be a subset of a defined universe of individually definite and performable actions.

(This is a condensation of a more complete definition by Knuth (I968 pp 4-6). It is believed that the shorter version suffices for this study. The definition includes computer source code and flowcharts (see below) as examples of media by which an algorithm may be communicated. A special type of algorithm is the sequential algorithm in which the set of actions is presented serially - in contrast, say to structured algorithms which are arranged hierarchically.)

Flowchart: Sequential algorithm in which the sequence of actions is shown by writing the actions within geometrically shaped boxes such as



and linking the boxes with arrows to show the order.

Coding: the act of converting an algorithm into a form that can be input to the computer and recognized by the compiler.

(The required form will necessarily be character-strings, which to be recognized by the compiler must obey the special rules of the programming language being used. In this manner the programmer produces source code which the compiler converts into object code for execution. In this study, the word "code" will imply source code.)

Bug: Programming error, of either of two kinds:

Syntax Bug: Violation of the formal rules of the programming language being used.

Logic Bug: Failure of a program to perform the required task, even when the syntax is correct.

(Syntax bugs make the program incomprehensible to the compiler and prevent it from running at all. Syntax rules are language-dependent: perfectly syntactical FORTRAN programs would be nonsense to a COBOL compiler.)

The Classroom Programming Environment

Since programs and programmers must be examined in relation to the environment in which they operate, consideration was given to the situation of the pupil undertaking an elementary programming course in a typical New Zealand classroom. Organizationally, he belongs to a class group numering between 20 and 40, and therefore has to rely mainly upon his own insights or discussion with skill peers (other pupils) rather than with a skill model (the teacher) in developing his programs. He almost certainly makes use of a batch-processing computer system, in which programs and data are prepared away from the computer; completed jobs are fed into the computer at one go (a batch) and the student waits anything from some hours to several days to see what results he has. Other programming systems exist, notably conversational programming, which are educationally superior but require more expensive hardware than is likely to appear in many New Zealand schools.

The achievement of the student learning programming is affected by the content of his course and the preoccupations of his teacher, which are in turn influenced by prevailing attitudes to programming as a school subject. Thus the kind of project in which the pupil is most often involved is a job capable of being accomplished with a relatively short program that will probably be abandoned after one or

two runs. The task is normally set by the teacher, by whom the solution is also judged. The criterion for judgement is usually "correctness", i.e. freedom from bugs; program efficiency is not usually considered in elementary courses. It is also unlikely that systematic debugging techniques or aids are available to him, other than the diagnostic facility of the compiler which can identify syntax errors though not logic ones.

Since many teachers are concerned with the more general aspects of programming, there is often some pressure on the student to document his algorithm in some uncoded form (often a flowchart) before proceeding to the coding phase. Under these circumstances, the following was assumed to be a valid description of programming activity by students in an elementary programming course for a batch-processing system, in an ordinary New Zealand classroom environment:

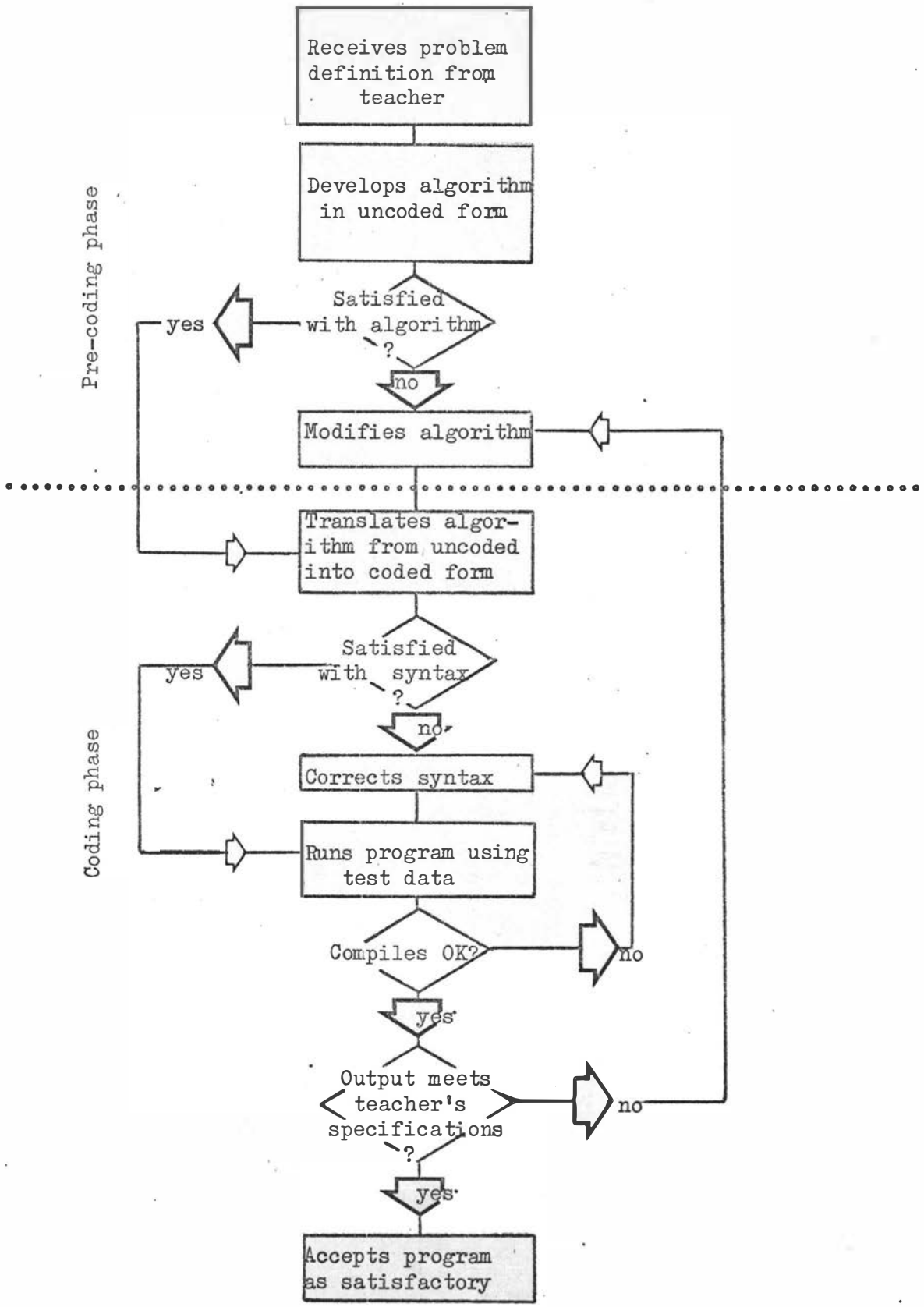


Figure 2: **Model of Student Behaviour in Undertaking Classroom Programming**

It was also assumed that the following abilities exist in the population and can be measured by suitable pencil-and-paper tests:

Algorithmic Skill: Ability to develop an algorithm in uncoded form for a given task, using elements chosen from a given universe of individually definite and performable actions.

(Note: the "uncoded form" may be a flowchart)

Coding Knowledge: Ability to translate an algorithm accurately into syntactically correct computer instructions in a given programming language.

Hypotheses Investigated in this Study

The hypotheses for this study arose from the four items of folk-wisdom with which the Introduction opened.

Skill at coding is not related to skill at developing algorithms, and vice versa. To investigate the validity of this statement at the superficial level, all that would be required would be a test of significance for a zeroth-order correlation between measures of the two skills. But this correlation would tend to be inflated by general intelligence and testwiseness, both of which affect scores on any pencil-and-paper test. There was greater interest, then, in the question of whether a relation still existed between scores on tests of algorithmic skill and coding knowledge after removing the effect of performance on a test of general intelligence.

Personality is just as important in predicting success at programming as are past attainments. This statement raises the whole question of prediction of programming performance. In this study, the tests of algorithmic skill and coding knowledge would be used as criterion measures and therefore would become the dependent variables in the prediction procedure. The linear regression model had been used in previous prediction studies (e.g. C.A. Alspaugh, 1972) and was therefore judged most appropriate for this one; by

including personality variables with attainment variables as predictors, the relative contribution of each type would be examined.

Attainment in mathematics and attainment in programming are unrelated to each other. Like the first statement above, this could be examined at the superficial level, but also like it, could be better studied when the effect of performance on a test of general intelligence was removed. Even more suggestive would be a study of the two measures of programming achievement against individual components of mathematical attainment, to see if patterns emerged that would lead to more detailed understanding of the relationship between the two subject areas.

The instructor's strategy makes no difference to the student's attainment. One of the more persistent arguments among teachers of programming is whether coding should be taught before the techniques of developing uncoded algorithms, or vice versa. This question could be tested experimentally by exposing one sample group to one method, and another to the other. At the end of the process the two groups could be compared to see whether there was any significant difference in performance on the programming attainment tests and if so, in which direction. A control group, which would receive no programming instruction whatever, could furthermore be used to see whether programming instruction had actually improved the performance of the two taught groups on the test of algorithmic skills, or whether maturity and environmental factors alone would account for attainment in the test.

Expressed in null form, the hypotheses to be tested were thus

I. That there is no relation between performance on a test of algorithic skill and a test of coding knowledge, when the effect of performance on a test of general intelligence is eliminated.

2.I That there is no relation between performance on a test of mathematical attainment and a test of

algorithmic skill, when the effect of performance on a test of general intelligence is eliminated.

2.2 That there is no relation between performance on a test of mathematical attainment and a test of coding knowledge, when the effect of performance on a test of general intelligence is eliminated.

3.1 That a regression model will not significantly predict performance on a test of algorithmic skill from past academic performance and a range of personality measures.

3.2 That a regression model will not significantly predict performance on a test of coding knowledge from past academic performance and a range of personality measures.

4. That there is no difference in performance on either a test of algorithmic skill or of coding knowledge between children taught

(a) flowcharting before coding,

(b) coding before flowcharting,

(c) no programming whatever.

CHAPTER III

METHODS AND MATERIALS

Of the hypotheses to be examined, four (nos I to 2.3 on pp 21,22 above) required concurrent measurement of attainment in mathematics, algorithmic skill and coding knowledge at the conclusion of a programming course, so that the relationships among these measures might be investigated. The remaining hypotheses concern the prediction of achievement in manipulating algorithms and in coding on the one hand, and *post hoc* comparison of different teaching strategies on the other.

The common elements in all these investigations were Algorithmic Skill and Coding Knowledge. Because suitable instruments were not available for measuring these, new ones had to be constructed for the purpose; their development and evaluation is described in detail in the next chapter.

It was also decided that another common element in the investigations should be the content of the course presented to the students. Comments by J.W. Alspaugh (1971), Weinberg (1971) and others, as well as common experience suggest that course content and particularly the programming language taught, is a significant variable in evaluative studies. Because the present study was not concerned with the effect of this variable, it was decided to control for it by holding it constant. The choice of instructional materials was thus relevant to the present research, but as no theoretical reason was found for preferring one set of materials over another, the choice was made on practical grounds.

Choice of a Programming Course

Since batch-processing systems were considered to be the type most likely to be implemented in New Zealand schools, and since the widest applicability was sought for the study, internationally-distributed courses on

elementary Fortran programming,

elementary COBOL programming,

CESIL programming, and

batch-processing BASIC programming

were considered. Of these, the Computer Education in Schools course, Computer Studies I (ICL/CES, 1971) was selected because its materials are suitable for 12-year-olds, its textbook and teaching aids are internationally distributed and computing facilities capable of supporting the course are readily obtainable anywhere in New Zealand. Only one programming language is used in Computer Studies I - the Computer Education in Schools Instructional Language (CESIL). Used solely for educational, not professional, programming, CESIL resembles the well-known SAMOS, also BAL which was used by C.A. Alspaugh in her prediction study (see above, p. 11) and the code handled by Clutterham's compiler (above, p. 8). CESIL is a low-level code for programming a simulated single-accumulator computer by means of instructions taking the form

(label) (operator) (operand)

the rules for which are described in detail in the next chapter.

As well as discussing how to code programs in CESIL, Computer Studies I also treats the development of uncoded algorithms in some detail, expressing them in flowchart form. There is also material on computer hardware and applications, which for present purposes is of peripheral interest only.

For the studies of concurrent achievement, the course teaching followed Computer Studies I exactly. This was done during 1973. In 1974, when the comparative study of teaching sequences was undertaken, the same material was reorganized and slightly augmented to create the necessary pair of alternative courses. In these courses, which are detailed in Appendix A, one has the sections on CESIL coding before the sections on flowcharting, and the other has the identical sections in reverse order. In all cases,

a four-week period of undirected practical work was included between the conclusion of formal instruction and the commencement of the testing programme, lest the tests reflect merely short-term recall of the most recent learning.

Like course content, teacher style was a variable which though not a subject of this investigation, was considered to be significant to the outcomes of a programming course. As with course content, it was decided to control for this variable by holding it constant: the same teacher (who was also the researcher) taught all the courses.

Other attributes that were recognized as intervening variables were general intelligence, age, and (possibly) sex. Since these were not controlled for, their distribution in the research samples was examined and compared with national norms. The selection and characteristics of those samples were as follows.

The Samples Used in the Study

The universe of interest for the present study was all Form 2 children in New Zealand intermediate schools. As the research design required weekly contact between a single teacher and all sections of the sample, it was necessary for the population sampled to be contained within one geographical area: the area selected was the Hawkes Bay province.

The population thus being identified as the Form 2 pupils of Hawkes Bay intermediate schools, all five such schools were invited to participate in the research programme. Of these five, three elected to join the programme: the other two declined on the grounds that they were already involved in other experimental programmes. As a check against bias resulting from the omission of these two schools, the opinion of the District Senior Inspector of Primary Schools for Hawkes Bay, Mr. A. Lawrence, was sought; he stated in a formal

memorandum that in his opinion, the pupils of the participating schools were indeed representative of all intermediate school pupils in Hawkes Bay. Each of the participating schools had, within its normal timetable, periods when Form 2 pupils were in "unstreamed" groups of the required size (20 - 40).

After careful enquiry, it was decided that one of these groups in each school could be regarded as a random sample of the Form 2 children attending that school. In this way a sample of 96 subjects was obtained, in three subgroups. This was the sample used in 1973 to examine the relationship among Algorithmic Skill, Coding Knowledge and mathematical attainment. Ages, Otis IQ scores from the schools' records, and sex ratios were distributed as shown in Table 3:

Table 3: 1973 Descriptive Statistics from Sample of 96 Subjects Used in Correlational Study and for All New Zealand Form 2 Pupils: Age, Otis IQ (Means and Standard Deviations) and Sex Ratios.

	Sample		All Form 2 Pupils, 1973	
	Mean	s.d	Mean	s.d
Age in years at Jan. 1	12.4	0.45	12.4[a]	0.57[a]
Otis IQ	101	15	100[b]	15[b]
Boy:Girl Ratio	54:46		51:49	

One of the three subgroups of this sample was used in late 1973 to examine the test-retest performance of the Coding Knowledge Test. The characteristics of this sub-sample are shown in Table 4:

a Source: NZ Department of Education, Private memorandum
b Source: Manual for the Otis Intermediate Test (NZCER, 1969

Table 4: Descriptive Statistics for Sample of 36 Subjects Used in Test-Retest Study of Coding Knowledge Test: Age, Otis I_Q (Means and Standard Deviations) and Sex Ratio.

	Mean	s.d
Age in Years at Jan. 1	12.4	0.36
Otis I_Q	101	14
Boy:Girl Ratio	70:30	

For the 1974 study of different teaching strategies, it was required to control for the group of variables, "school environment", and therefore it is necessary to carry out that experiment within a single school. Of the three schools participating in the experiment, Wycliffe Intermediate School was arbitrarily chosen to provide the experimental samples. Three unstreamed groups were provided by the Principal: these numbered 36 (group AC), 31 (group CA) and 32 (group CONTROL). Characteristics of the groups were as shown in Table 5:

Table 5: Descriptive Statistics for Sample Groups of Subjects Used in Study of Different Teaching Strategies: Age, Otis I_Q (Means and Standard Deviations) and Sex Ratios.

	Group AC		Group CA		Group CONTROL	
	Mean	s.d	Mean	s.d	Mean	s.d
Age in Years at Jan. 1	12.4	0.59	12.4	0.40	12.5	0.43
Otis IQ	100	11	98	12	99	11
Boy:Girl Ratio	49:51		45:55		47.53	

For the 1974 prediction study, groups AC and CA above (i.e. the two groups who actually received instruction in programming) were combined into one sample of 67 subjects. Table 6 shows the characteristics of the sample so produced:

Table 6: 1974 Descriptive Statistics for Sample of 67 Subjects Used in Prediction Study and for all New Zealand Form 2 Pupils: Age, Otis IQ (Means and Standard Deviations) and Sex Ratios.

	Sample		All Form 2 Pupils, 1974	
	Mean	s.d	Mean	s.d
Age in Years at Jan. 1	12.4	0.51	12.4[a]	0.56[a]
Otis IQ	101	15	100[b]	15[b]
Boy:Girl ratio	47:53		51:49	

The same sample was also used to determine the parallel-form reliability of the Coding Knowledge Test.

In 1975, a special project was undertaken to determine the reliability of the Algorithmic Skill Test. The sample chosen for the purpose was selected by the same method as the 1973 one (see above, p.26), contained 93 subjects, and had the characteristics shown in Table 7:

Table 7: Descriptive Statistics for Sample of 93 Subjects Used in Determination of Reliability of Algorithmic Skill Test: Age, Otis IQ (Means and Standard Deviations) and Sex Ratio.

	Mean	s.d
Age in Years at Jan. 1	12.4	0.45
Otis IQ	98	15
Boy:Girl ratio	40:60	

[a]Source: NZ Department of Education. Private memorandum

[b]Source: Manual for the Otis Intermediate Test (NZCER,1969)

Tests Chosen for the Study

The study of the relationship between mathematics attainment and attainments in algorithmic skill and programming knowledge required a mathematics test which not only provided a measure of overall attainment but also revealed an infrastructure of subscores in particular aspects of the subject. No such test existed in 1973 when the study was undertaken, and it was therefore decided to construct a special battery from available materials. The sections chosen were

Arithmetic
Numerical Relationships
Spatial Relationships
Logical Relationships

For the arithmetic section, the ACER Arithmetic Tests (ACER, undated) were chosen, as they are the standard tests of arithmetical skills in this country. There are four tests (addition, subtraction, multiplication, division), standardized by Fieldhouse (1956) for New Zealand conditions. From Fieldhouse norms, raw scores on each part were converted to decile scores, which in turn were subjected to a standard normalizing transformation. The mean of the four normalized scores - the "Mean Normalized Arithmetic Score" was used as an overall measure of arithmetical skill.

To cover the other sections, items were selected from Multiple Choice Tests in Mathematics for Junior Forms (Spence, 1973). This series of tests had been designed for Form 3 students, but in view of the integrated nature of the mathematics syllabus for Forms 1 - 4, it was considered that the easier items from the Form 3 test would be suitable for Form 2 children. The advantages of using items from this source were that they had been developed for New Zealand conditions, were arranged in topic groupings, and had been tested for difficulty and discrimination within those groupings. (Though the indices of difficulty and discrimination have not been published, they were determined by the researcher when the Form 3 tests were constructed.) The items selected

for the Form 2 test were those within the required topic groupings that had the lowest difficulty indices. In this way the Mathematics Attainment Test was constructed, yielding sub-scores in Numerical Relationships (items 1 - 10), Spatial Relationships (items 21 - 30) and Logical Relationships (items 11 - 20). This test is shown in Appendix B.

The comparative study of different teaching strategies required no instrument not already mentioned, but the prediction study needed measures of various personality factors and of relevant academic attainment. Treating the latter question first, it is unclear what attainments, other than mathematical, may be relevant. In this study, vocabulary skills were investigated as possible predictors, particularly of coding knowledge which demands the understanding of special symbols in a very exact way. Furthermore, when the programming language involves context-free statements as does COBOL, a test measuring comprehension of words out of context was considered to be most suitable. For this reason, the Progressive Attainment Test in Vocabulary (Elley and Reid, 1969), was used. For mathematics the officially-endorsed Progressive Attainment Test in Mathematics (Reid and Hughes, 1974) was selected to measure personality variables, the widely-used High School Personality Questionnaire (Cattell and Cattell, 1968) was adopted. This test yields fourteen factor scores, the factors having the labels:

A Reserved/Warmhearted
B Less Intelligent/More Intelligent
C Affected by Feelings/Emotionally Stable
D Undemonstrative/Excitable
E Obedient/Assertive
F Sober/Enthusiastic
G Disregards Rules/Conscientious
H Shy/Adventurous
I Tough-minded/Tender-minded
J Zestful/Circumspect Individualism
O Self-assured/Apprehensive
Q2 Sociably Group-dependent/Self-sufficient
Q3 Uncontrolled/Controlled
Q4 Relaxed/Tense

The Structure of the Study

The present investigation concerned some of the properties of the abilities called Algorithmic Skill and Coding Knowledge. In considering these abilities, attention was given both to their internal properties as they relate to each other and contribute to programming achievement, and to such relationships as might be found with phenomena beyond the sphere of programming.

Investigation of the internal properties first required means whereby the abilities might be measured. Chapter IV contains a detailed account of the way in which tests of Algorithmic Skill and Coding Knowledge were constructed and tested to determine their characteristics. With these tests, a correlational experiment was mounted to investigate the validity of the assertion that pre-coding skills are independent of coding skills.

To examine relationships between the two abilities and external phenomena, detailed consideration was given to the relationship with attainment in the subject mathematics, by means of a concurrent correlational survey between Algorithmic Skill, Coding Knowledge and a range of aspects of mathematical attainment. Mathematical attainment was also used as one of the predictors in a regression analysis designed to test the capacity of academic and personality variables to predict end-of-course attainment in a programming course. The full set of predictors consisted of raw scores on tests of mathematics attainment, attainment in English vocabulary, personality factors, attainment in mechanical arithmetic skills and a dichotomous variable indicating which of two course presentations the subject had been exposed to. These variables were combined linearly to maximally predict each of Algorithmic Skill and Programming Knowledge.

Another external phenomenon that might or might not affect Algorithmic Skill and Coding Knowledge was investigated: different strategies used by the instructor

in presenting a programming course. Three sample groups were selected, one to learn flowcharting before coding, one to learn coding before flowcharting, and a third - a control group - to receive no instruction in programming whatever. By comparing the achievements of these groups at the end of the period of instruction, it was possible to determine first whether instruction genuinely improved performance, and second whether the choice of strategy made any difference. A one-way analysis of variance was performed on end-of-course scores in Algorithmic Skill and Coding Knowledge, to test differences among the two taught and one untaught groups. As elsewhere throughout this investigation, the level of significance was set at 0.05.

How the Study Developed

The entire investigation was carried out during the four years 1972 - 1975. 1972 was devoted to development and trials of methods and materials. By the end of that year, the Computer Studies 1 course had been tested, and the instruments measuring algorithmic skill and coding knowledge had been constructed, analysed and revised. During 1973 - 5, opportunities were taken to determine the empirical characteristics of these new tests. The special test of mathematical attainment was also prepared in 1972 for use in 1973.

At the end of 1973, after the standard Computer Studies 1 course had been taught, the testing programme for the concurrent study of relationships between programming and mathematical attainment was undertaken. These data were subsequently analysed by correlational methods, using zero-and first-order partial correlation coefficients between the relevant variables.

Two projects, running in parallel, were undertaken in 1974. The first of these was the prediction study, for which some data on previous academic achievement were obtained from school records. To gather the remaining data needed for the study, further attainment

tests and the personality test were administered, all before the course of instruction commenced. These data provided the information from which it was hoped to predict scores on the programming tests at the conclusion of the course. At the same time and with the same pupils, a second investigation was undertaken, comparing the effects of two different orders of presentation of the course material. One sample group was taught coding first, then the development of algorithms in uncoded form; the other group was taught in the opposite sequence. At the end of the process, both groups and a control sample that had had no programming instruction whatever, were tested to measure algorithmic skill. At the same time, the two groups that had had programming instruction were tested for coding knowledge. Comparison of the scores so obtained would permit evaluation of the two teaching strategies.

The flow diagram on the next page shows the development of the investigation, except for the 1975 project to determine the reliability of the Algorithmic Skill Test - this was omitted for greater clarity.

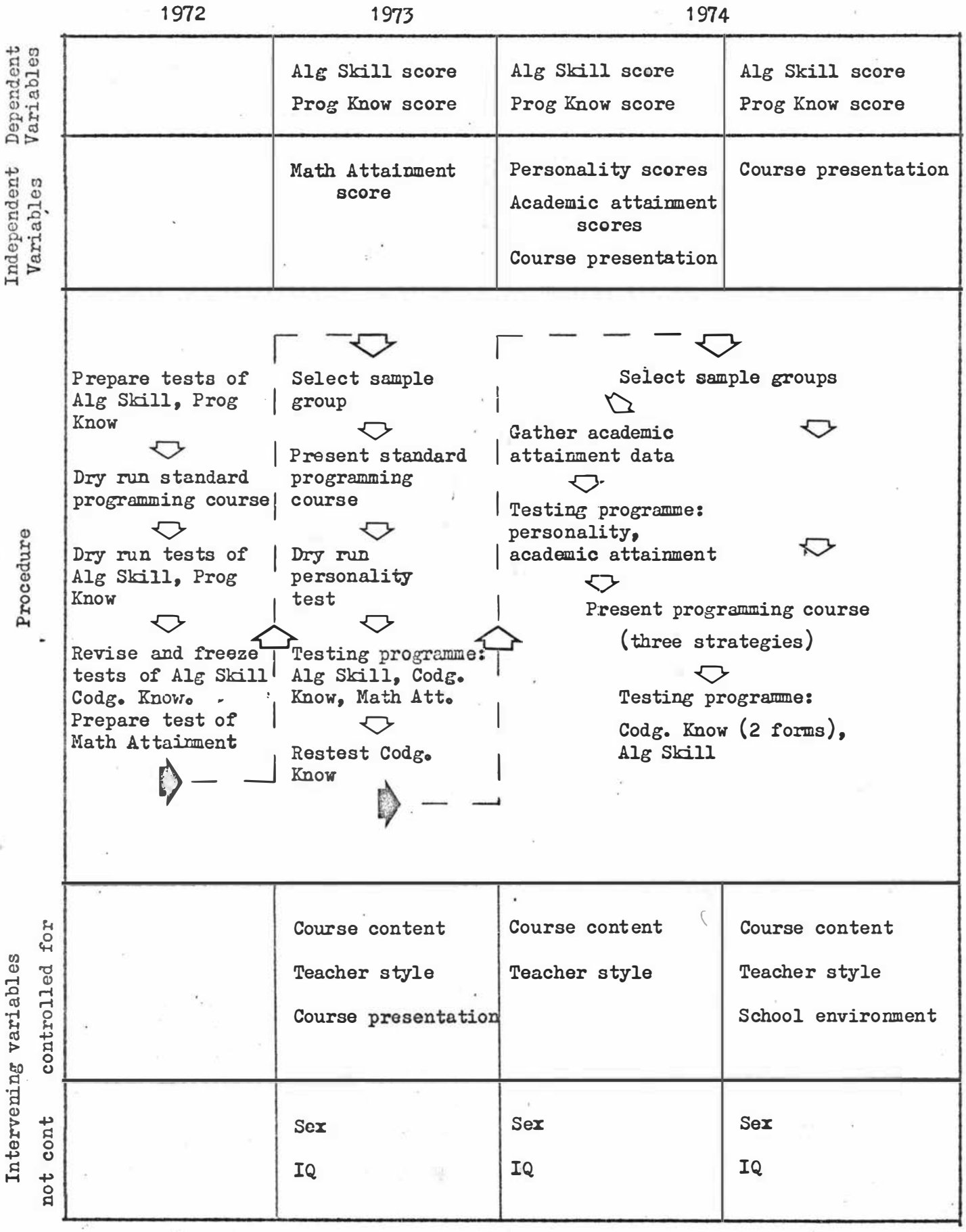


Figure 3: Flow Diagram showing Chronological Development of Study.

CHAPTER IV

THE DEVELOPMENT OF TESTS OF CODING KNOWLEDGE AND ALGORITHMIC SKILL

It has already been noted (above, p.11) that of the studies enumerated in the literature survey, only one used an objective test as the criterion of programming achievement. Howell et al., who undertook that study, used the U.S. Department of Census Robot Test, which, they say

"provides a sample of programming activity analogous to problem solving on a large-scale digital computer. The problems in the Robot Test require an individual to write instructions in a symbolic language for a 'Robot Clerk'... the test, though in its eighth revision, is still in a developmental stage and is not well standardized."

An important aim of the present study was to prepare objective tests of Coding knowledge and Algorithmic Skill. Since Computer Studies 1 uses only CESIL as a programming language and flowcharts for expressing uncoded algorithms, the tests were of CESIL and of flowcharting.

Construction of the Coding Knowledge Test

To construct the test of CESIL coding knowledge, the language was surveyed in the light of the model of programming behaviour (above, p.19). The task in the coding phase was seen to be two-fold:

1. to translate the uncoded algorithm into code.
2. to ensure that the syntax of the code was valid.

Treating the second part of the measurement problem first, it was required to construct test items that would measure a subject's ability to distinguish between valid and invalid code. First, a survey of CESIL syntax was undertaken so that a universe of all possible items might be conceived. The items actually chosen were seen as a sample of this universe.

Because of the simplicity of CESIL syntax, a complete analysis was possible. The form of each instruction is

(label) (operator) (operand)

and the rules for constructing valid instructions are:

1. The label may or may not be used in a given statement. If used, it must obey the rules for an identifier (see Computer Studies 1, p.17).

2. An operator must appear in the second position of the statement, and must be one of the twelve CESIL verbs (see Computer Studies 1, pp.275-6).

3. The operand must be one of : an identifier
   a space
   a numeric literal (see Computer Studies p.71 for the rules for literals.)

4. The operand must be of a type compatible with the operator. The only permitted combinations are

Operator	Permitted Operands
HALT, IN, OUT	Space
LOAD, ADD, SUBTRACT, MULTIPLY, DIVIDE	Identifier or literal
STORE, JUMP, JNEG, JZERO	Identifier

It was decided to test subjects' knowledge of the syntax rules of CESIL by measuring their ability to discriminate between valid and invalid instructions. Items were constructed in multiple-choice form with four options, of which three would be syntactically invalid and one valid. The subject would receive one point for correctly identifying the valid statement in the set, but suffer no penalty for incorrect choices.

Label syntax would not be tested directly, as identifiers (the only valid type of label) could be covered through operands.

For each of the tests six sets of four unlabelled statements were generated, having these characteristics:

	Form A	Form B
Valid statements :	6	6
Statements violating rule 2 :	1	1
Statements violating rule 3 :	5	4
Statements violating rule 4 :	10	10
Statements violating both rules 3 and 4 :	2	3
	24	24

The resulting six multiple-choice items chosen for forms A and B of the CESIL syntax part of the Coding Knowledge Test may be examined in Appendix C.

Turning to the problem of testing the ability to translate uncoded algorithms into CESIL code, it was again necessary to examine the features of this particular programming language. A feature of CESIL is the fact that the meaning of each instruction is entirely context-free: it does not depend on statements in other parts of the program. In this, CESIL differs from, say, COBOL in which instructions of the PERFORM variety can only be interpreted by referring to other parts of the program.

Because of the context-freedom of the meanings of CESIL instructions, a subject's ability to translate from flowchart to CESIL code would depend solely on his knowledge of each separate type of CESIL instruction[1]. As already noted, there is a finite list of valid CESIL instructions:

Operator	Operand
HALT, IN, OUT	Space
LOAD, ADD, SUBTRACT, MULTIPLY, DIVIDE	Indentifier or label
STORE, JUMP, JNEG, JZERO	Identifier

The presence of a syntactically valid label does not affect the meaning of an instruction, so it was concluded that the universe of meanings for CESIL instructions contains

$$3 + 2\times5 + 4 = 17 \text{ classes.}$$

In the CESIL Coding Knowledge test, subjects were required to demonstrate comprehension of CESIL code by predicting the effect on given data of instructions drawn from each of the 17 classes. It was decided to imbed each instruction in a program of minimal length; input data were provided and subjects were required to predict the resulting output. If a subject predicted the same characters, in the same order, as the computer itself would have output, he was deemed to have demonstrated understanding of that type of instruction.

---

1 The justification for this statement relies upon the assumption that the flowchart in question is of the type referred to in Computer Studies 1 as a "computer flowchart" (p.96). In terms of the definitions on p.16 above, a computer flowchart may be described as an algorithm in flowchart form for which the defined universe of actions is those actions capable of being executed in a single step by the (simulated) computer.

Such a "behavioural" viewpoint nevertheless created difficulties. Every one of the programs used as a test item must have input, output and program termination instructions - IN, OUT and HALT. To make the items work, it was essential that every subject understand the exact significance of these three instructions, which in turn implied that such understanding could not be tested. Further, to create programs containing certain instructions of the type

(operator) (identifier)

it was necessary to include a

STORE (identifier)

instruction preceding it, even if the program was to be "of minimal length", and it is debatable whether this requirement placed too great an emphasis on mastery of the STORE type instruction.

The first (Mark 1) version of the test contained one item of the type described for each of the 17 classes of instruction except IN, OUT and HALT, which were carefully explained in the introduction to the test (see Appendix C). It was considered, however, that equal sampling from all 14 remaining classes tended to over-value mastery of the LOAD, ADD, SUBTRACT, MULTIPLY and DIVIDE type instructions, each of which was twice represented, once in its form with an identifier as operand, and once with a literal. For the Mark 2 version, used for all work after 1972, the items having literals as operands were combined into a single item. The effect on test homogeneity of this condensation was tested by examining correlations between item scores and test total:

Table 8: Correlation Coefficients between Item Scores and Test Totals for Coding Knowledge Test Marks 1 and 2, and Changes Introduced between Marks, by Item.

Item Mk1	Item - test correlation Mark 1,1972	Changes introduced in constructing Mark 2 test booklet	Became Item Mk 2	Item-test correlation Mk 2, 1973
1 (p.3)	+0.66	None	1 (p.3)	
2 (p.4)	+0.54	None	2 (p.4)	
3 (pp 5-6)	+0.54	None	3 (p.5)	
4 (p.7)	+0.49	None	4 (p.6)	
5 (p.8)	+0.07	None	5 (p.7)	
6 (p.9)	+0.43	None	6 (p.8)	
7 (pp 10-13)	+0.74	pp10-12,involving ADD,SUBTRACT,MULTIPLY,combined into a single program. B13 (DIVIDE) dropped entirely. Score for item changed from 4 points to 1 point.	7 (p.9)	+0.68
8 (p.14)	+0.56	None	8 (p.10)	
9 (pp15-16)	+0.67	None	9 (p.11)	
10 (p.18)	+0.80	None	10 (p.13)	

Notes:

1. Pearson product-moment correlation coefficients reported.
2. The procedure of administration and scoring remained unchanged from Mark 1 to Mark 2.

Validity of the Coding Knowledge Test

It was not possible to consider the construct validity of the Coding Knowledge Test, because reliable information about the construct was lacking. Weinberg's (1971) general discussion, for example, was based on informal observation rather than empirical research. Content validity, therefore, was all that could be established; the evidence for content

validity was in fact the method of the test's construction, as described above (pp. 35 - 39). Inasmuch as the Coding Knowledge Test represents an unbiassed sample from the universe of relevant behaviours, the test possesses substantial content validity.

Reliability of the Coding Knowledge Test

The test-retest behaviour of the Coding Knowledge Test was investigated at the end of 1973, using the small sample of 36 subjects (see above, p. 27). On the first administration of the test, immediately at the end of the teaching programme, the mean score was 8.0, while on the second administration three weeks later it was 7.8. This demonstrated that no significant overall change in performance level had occurred between the two administrations, and therefore the test-retest reliability could with justification be determined by the correlation between corresponding pairs of scores.

Pearson product-moment correlation coefficients between scores on test and retest were taken, yielding the following values for 31 cases:

CESIL Functions	0.87
CESIL Syntax	0.52
Test Total	0.81

Comparing Forms A and B of the test at the end of 1974, using the sample of 66 subjects (see above, p. 28), produced the following means and standard deviations:

	Form A	Form B
Mean	8.0	8.4
Std.dev.	4.0	4.3

It was thus demonstrated that taken overall, the two forms of the test could be regarded as equivalent, and that the parallel-form reliability could validly be determined by the correlation between scores on the two forms.

The Pearson product-moment correlation coefficients between scores on Part A and Part B took the following values:

CESIL Functions	0.90
CESIL Syntax	0.77
Test Total	0.93

These values, when taken together with the above values for the means and standard deviations of scores on the two forms, show not only that the reliability of the Coding Knowledge Test is high, but also that the two forms may be regarded as interchangeable.

Finally, for completeness, a Pearson correlation coefficient was computed for the intercorrelation between CESIL Functions score and CESIL Syntax score. For the 1973 sample of 96 subjects (see above, p.26), this coefficient had a value of 0.68.

Construction of the Algorithmic Skill Test

Preparation of a test of Algorithmic Skill was helped by the fact that flowcharting tests are used in measuring programmer aptitude. Three of the five tests listed on p.13 above contain subtests on flowcharting, the best documented being the "diagramming" part of the Computer Programmer Aptitude Battery (CPAB) (Palermo, 1964).

It was decided to construct a junior version of the CPAB diagramming test. Like the CPAB, the Algorithmic Skills Test would be speeded. Although Wolfe (1971) has contended that speeded tests of skills related to programming are inappropriate, it was considered that stronger arguments existed for retaining the element of speed. It was believed that the subject who laboured through an item step-by-step was demonstrating an inferior level of skill to the one who achieved global perception of the algorithm and thereby described its effect more rapidly. The imposition of a time limit ensured a higher score for the more sophisticated subject.

In the original 1972 version of the test, the time limit was set at 15 minutes, but for those subjects who finished more quickly, a record was kept of their completion times. Detailed examination of this record led to a decision to reduce the time limit to 12 minutes, which would prevent nearly all subjects from completing the test. This was considered to be consistent with the aims of the test, and was applied in all administrations during the main period of the investigation (1973 and 1974).

In the writing of the test items, the special nature of the target population was borne in mind. First, it was considered useful to have items that could be graded right/partly-right/wrong, rather than simply right/wrong, and this required a format revealing the process by which the subject reached his conclusion. Second, items were sought that would permit the language used on the flowcharts to be extremely brief but unambiguous to twelve-year-olds, some of whom would be disadvantaged readers. In the same way, problems were sought that would not make demands upon the subject's level of numeracy. The result was a set of items in which each flowchart referred to the construction of a geometric pattern (see Appendix D).

Three types of item were used in the Algorithmic Skill Test. One type (items 1, 2, 6) involved simple sequences of operations without conditional branches or variable elements. Loops and variables were reserved for the second category (items 3,4,5,7,9). Both these types of item required the subject to interpret a flowchart and perform the task it specified; a third type of item required him to build parts of a flowchart by completing missing portions in a way that ensured that a prescribed pattern would emerge. The items in this category (nos. 8,10) were presented in multiple-choice format, and most closely resembled the exact items of the CPAB diagramming test.

Between 1972 and 1973, the scoring procedure for two test items was revised to improve their objectivity. The effect on test homogeneity was tested by examining correlations between item scores and test total, reported in Table 9 below.

Table 9: Correlation Coefficients between Item Scores and Test Total for Algorithmic Skill Test in and after 1972, and Changes Introduced in Revision of Procedure, by Item.

Item No.	Item - test correlation 1972	Changes in procedure introduced after 1972	Item - test correlation 1973
1	+0.14	None	
2	+0.43	None	
3	+0.52	Requirement for 2 points made more rigorous	+0.51
4	+0.71	Requirements for 2 points made more rigorous	+0.47
5	+0.68	None	
6	+0.28	None	
7	+0.63	None	
8	+0.32	None	
9	+0.54	None	
10	+0.22	None	

Notes

1. Pearson product-moment correlation coefficients reported.
2. The content of the items remained unaltered (i.e. the Mark 1 version of the test booklet was used throughout).
3. Each item is scored 0, 1 or 2 points, as described in the marking scheme in Appendix D.

Validity of the Algorithmic Skill Test

Since the universe of behaviours relevant to Algorithmic Skill is so far ill-defined, it was not possible to establish the content validity of this test on the grounds of sampling as was done for the Coding Knowledge Test. Rather, it was necessary to relate the Algorithmic Skills Test to its pattern, the SPAB diagramming test, which has been shown (Palormo, 1967) to have predictive validity against the criterion of supervisor ratings. Some measure of validity - actually a mixture of content and construct validity was claimed on the grounds that the items of two tests require the same kind of performance at different levels of skill.

Reliability of the Algorithmic Skill Test

Among the 96 subjects tested in 1973, total scores on the Algorithmic Skills test were almost normally distributed:

Mean	9.6
Std.dev.	3.3
Kurtosis	-0.21
Skewness	-0.10

Inasmuch as the Algorithmic Skills test is speeded, split-half methods were inappropriate for estimating the test's reliability, but inasmuch as it was a mastery test, neither would test-retest methods yield a useful measure. In a study specially mounted in 1975, the test was administered to a sample of 93 subjects (see p.28), but without the time-limit. The resulting scores were analysed by the generalized Kuder-Richardson method, (see, for example, Ebel (1970), Mehrens and Lehmann (1973)) yielding a reliability coefficient of 0.75.

The tests of algorithmic skill and coding knowledge were ready in their final form by early 1973 when the first of the three main studies began, and remained unchanged throughout the rest of the investigation.

CHAPTER V

RESULTS

Composite Mathematics Attainment Score Defined

Four measures of mathematical attainment (see above, pp.29-30) were used in this study, and it was necessary for the purposes of the investigation that they be combined into a single overall measure. In an endeavour to find a suitable combination, The PAT Mathematics Test (Reid and Hughes, 1974a), then under construction, was considered. This test occupies a special position in mathematics instruction in New Zealand, having been built up on the opinions of mathematics teachers throughout the country. Because of the breadth of its base in teacher opinion, the PAT Mathematics Test can be regarded as the authoritative instrument for measuring mathematical attainment among New Zealand children. But because the PAT test was not available in 1973, and because the study of the relationships between programming and mathematical achievements required subscores on particular aspects of mathematics (which PAT Math does not provide), the test actually used was the one shown in Appendix B and discussed in Chapter III (pp.29-30) above. After careful consideration of the item description contained in the PAT Test's Manual (Reid and Hughes, 1974b), it was decided that a "Composite Mathematics Attainment Score" roughly equivalent to the PAT Math score could be arrived at by a linear combination of subscores, with these weighting coefficients:

Numerical Relationships Score	24
Spatial Relationships Score	12
Logical Relationships Score	8
Mean Normalized Arithmetic Score	10

It was this combination that was used as the measure of mathematics attainment in the 1973 correlational study.

Algorithmic Skill vs Coding Knowledge

To test the null hypothesis that there is no relation between performance on the Algorithmic Skill Test and on the Coding Knowledge Test when the effect or performance on the Otis IQ Test is removed, the partial correlation coefficient between the first two scores, factored by the third, was computed. The value obtained was 0.46 for 95 cases ($p \leq 0.001$). At the level of significance chosen for this study - 0.05 - this value justifies rejection of the null hypothesis under a two-tail test, if the three population distributions are assumed normal.

The zero-order correlation matrix, from which the above result and also those which follow in the next paragraph were derived, was as shown in Table 10 :

Table 10: Matrix of Simple Correlation Coefficients Obtained in 1973 Correlation Study: Mathematics Attainment Subscores by Programming Attainment Subscores and Otis IQ.

	Otis IQ	Alg.Skill	Codg.Know.
Alg.Skill	0.60		
Codg.Know.	0.56	0.64	
Comp.Math.	0.72	0.70	0.75

(95 cases)

Algorithmic Skill and Coding Knowledge vs Mathematical Attainment

First-order partial correlation coefficients (factored by Otis IQ) were extracted to examine the relationship of Composite Mathematics Attainment Score to Algorithmic Skill and Coding Knowledge. The values obtained were:

Alg. Skill vs Comp. Math. 0.48 (95 cases, $p \leq 0.001$)

Codg. Know. vs Comp.Math. 0.60 (95 cases, $p \leq 0.001$)

Because these coefficients were neither very high nor very low, the fine structure of the relationships was also examined to see whether particular correlations might exist that were markedly different from the general level. A matrix of first-order partial correlations factored by Otis IQ was obtained for subscores of the three measures, and is shown in Table 11 :

Table 11: Matrix of Partial Correlation Coefficients Factored by Otis IQ, Obtained in 1973 Correlation Study: Mathematics Attainment Subscores by Programming Attainment Subscores.

	CESIL Functions	CESIL Syntax	Alg. Skill
Number	0.48^{***}	0.38^{***}	0.39^{***}
Space	0.38^{***}	0.31^{***}	0.39^{***}
Logic	0.46^{***}	0.40^{***}	0.26^{**}
Arithmetic	0.43^{***}	0.22^{*}	0.36^{***}
	(95 cases)		

$^{***}p \leq 0.001$ $^{**}p \leq 0.01$ $^{*}p \leq 0.05$

All the coefficients took values in the same general range, except those for Arithmetic vs CESIL Syntax and Logic vs Algorithmic Skill, which, though significant at the 0.05 level chosen for the investigation, were markedly lower than the other coefficients in the matrix.

The partial correlation coefficients were derived from the zero-order correlation marix shown in Table 12:

Table 12: Matrix of Simple Correlation Coefficients Obtained in 1973 Correlation Study: Mathematics Attainment Subscores and Programming Attainment Subscores.

	Otis	C.Funct.	C.Synt.	Alg. Skill.	Number	Space	Logic
C.Funct.	0.55						
C.Synt.	0.48	0.68					
Alg. Skill	0.60	0.61	0.56				
Number	0.70	0.67	0.57	0.65			
Space	0.53	0.56	0.48	0.58	0.61		
Logic	0.60	0.63	0.57	0.53	0.63	0.65	
Arith.	0.56	0.61	0.43	0.57	0.65	0.52	0.57

Consideration of the above zero-order matrix casts further light on the two lowest of the partial coefficients of Table 11 . The weak partial correlation between Arithmetic and CESIL Syntax clearly results from a weak simple correlation. For an explanation of the low partial correlation between Logic and Algorithmic Skill, however, it is necessary to note not just the fairly low simple correlation coefficient between the two, but also the relatively high correlation of each of them with Otis IQ which was the factoring variable.

Prediction of Programming Attainments

The regression study was designed to investigate the relative importance of academic attainment and personality factors in predicting the two dimensions of programming attainment. The method used was that of stepwise multiple regression, on three academic predictors (PAT Mathematics Test score, PAT Vocabulary Test score and Mean Normalised Arithmetic Score) and

fourteen personality predictors (the 14 factor scores of the HSPQ. The 66 subjects of the 1974 sample were used for the study. To start with, the dependent variable was the Algorithmic Skill Test score.

In the stepwise regression procedure, variables are introduced one at a time into the equation. The SPSS program permits the user to force certain variables in before others; this option was used to perform the regression analysis twice over, once with the academic predictor variables forced in before the personality ones, and once with the personality variables before the academic. Because the sample was made up of two subsample groups which had received different instruction, a dummy "treatment" variable was also introduced among the predictors, and was given the value +1 for each member of one group, -1 for the other. By introducing this variable after all the other predictors, a check was kept on the proportion of the variance of the dependent variable that was explainable by treatment effects.

Table 13 summarizes the results of the regression analysis when the academic predictors were forced into the equation before the personality variables. Two stages are shown: first, when only the three academic predictors had been introduced, and second, at the end of the process when all variables were in the equation. To show the proportion of the variance of the dependent variable explained at each stage, the square of the multiple correlation coefficient corresponding to each set is recorded, and also the values of the beta coefficients. Table 14 shows the corresponding results obtained when the personality variables were introduced before the academic ones.

Table 13: Stepwise Multiple Regression Analysis Using Algorithmic Skill Test Score as Dependent Variable, Introducing Academic Attainment Variables before Personality Variables: with Academic Predictors Only, and with All Variables in Equation; Beta Coefficients and Multiple R^2.

Variables in the Equation

Set	Name of Variable	Beta	Multiple R^2
Academic Predictors Only	PAT Vocabulary	0.53	
	PAT Mathematics	0.23	
	Mean Norm. Arithmetic	0.04	0.51
All Predictors	PAT Vocabulary	0.39	
	PAT Mathematics	0.28	
	Mean Norm. Arithmetic	0.03	
	HSPQ Factor E	-0.20	
	HSPQ Factor J	0.25	
	HSPQ Factor D	-0.11	
	HSPQ Factor Q4	-0.09	
	HSPQ Factor Q3	0.04	
	HSPQ Factor I	-0.11	
	HSPQ Factor C	-0.07	
	HSPQ Factor O	-0.07	
	HSPQ Factor B	-0.02	
	HSPQ Factor A	-0.03	
	HSPQ Factor G	0.00	
	HSPQ Factor Q2	-0.06	
	HSPQ Factor F	-0.10	
	Treatment	0.17	
	HSPQ Factor H	-0.02	0.65

Notes: 1. N = 66

2. HSPQ Factor O was not brought into the equation because its F-level fell below 0.01, the level prescribed by the SPSS program (see Nie, Bent and Hull, 1970, pp. 174 - 195).

3. The low value of beta associated with HSPQ Factor B (intelligence) is explained by the presence in the equation of the academic predictors, all of which are heavily loaded on intelligence.

Table 14: Stepwise Multiple Regression Analysis Using Algorithmic Skill Test Score as Dependent Variable, Introducing Personality Variables before Academic Attainment Variables: with Personality Predictors Only, and with All Variables in Equation; Beta Coefficients and Multiple R^2.

Variables in the Equation

Set	Name of Variable	Beta	Multiple R^2
Personality Predictors Only	HSPQ Factor B	0.19	
	HSPQ Factor Q3	0.19	
	HSPQ Factor J	0.33	
	HSPQ Factor C	-0.10	
	HSPQ Factor O	-0.10	
	HSPQ Factor E	-0.24	
	HSPQ Factor I	-0.27	
	HSPQ Factor D	-0.18	
	HSPQ Factor G	0.10	
	HSPQ Factor F	-0.13	
	HSPQ Factor Q2	-0.09	
	HSPQ Factor A	0.07	
	HSPQ Factor H	-0.09	
	HSPQ Factor Q4	-0.05	0.38
All Predictors	HSPQ Factor B	-0.02	
	HSPQ Factor Q3	0.04	
	HSPQ Factor J	0.25	
	HSPQ Factor C	-0.07	
	HSPQ Factor O	-0.07	
	HSPQ Factor E	-0.20	
	HSPQ Factor I	-0.11	
	HSPQ Factor D	-0.11	
	HSPQ Factor G	0.00	
	HSPQ Factor F	-0.10	
	HSPQ Factor Q2	-0.06	
	HSPQ Factor A	-0.03	
	HSPQ Factor H	-0.02	
	HSPQ Factor Q4	-0.09	
	PAT Vocabulary	0.39	
	PAT Mathematics	0.28	
	Mean Norm. Arithmetic	0.03	
	Treatment	0.18	0.65

These results show the three academic achievement measures to be superior to the fourteen factor scores of the HSPQ in predicting Algorithmic Skill scores. They confirm C.A. Alspaugh's (1972) finding with older students, that academic background was a stronger predictor than personality variables.

The whole process was repeated using Coding Knowledge score as the dependent variable, using the same predictor variables organized and treated in the same way. Once again, the academic test scores were found to be the superior set of predictors. (see Tables 15 and 16):

Table 15: Stepwise Multiple Regression Analysis Using Coding Knowledge Test Score as Dependent Variable, Introducing Academic Attainment Variables before Personality Variables: with Academic Predictors Only, and with All Variables in Equation; Beta Coefficients and Multiple R^2.

Variables in the Equation

Set	Name of Variable	Beta	Multiple R^2
Academic Predictors Only	PAT Mathematics	0.49	
	Mean Norm. Arithmetic	0.13	
	PAT Vocabulary	0.13	0.43
All Predictors	PAT Mathematics	0.42	
	Mean Norm. Arithmetic	0.22	
	PAT Vocabulary	0.05	
	HSPQ Factor D	0.21	
	HSPQ Factor H	-0.13	
	HSPQ Factor J	0.25	
	HSPQ Factor C	-0.21	
	HSPQ Factor B	-0.08	
	HSPQ Factor Q3	0.16	
	HSPQ Factor Q2	-0.04	
	HSPQ Factor F	0.17	
	HSPQ Factor A	0.05	
	HSPQ Factor Q4	0.05	
	HSPQ Factor G	-0.02	
	HSPQ Factor I	-0.01	
	Treatment	-0.18	
	HSPQ Factor E	-0.06	0.63

Notes: 1. N = 66

2. HSPQ Factor O was not brought into the equation because its F-level fell below 0.01, the level prescribed by the SPSS program (see Nie, Bent and Hull, 1970, pp. 174 - 195).

Table 16: Stepwise Multiple Regression Analysis Using Coding Knowledge Test Score as Dependent Variable, Introducing Personality Variables before Academic Attainment Variables: with Personality Predictors Only, and with All Variables in Equation; Beta Coefficients and Multiple R^2.

Variables in the Equation

Set	Name of Variable	Beta	Multiple R^2
Personality Predictors Only	HSPQ Factor J	0.25	
	HSPQ Factor C	-0.27	
	HSPQ Factor D	0.24	
	HSPQ Factor Q3	0.16	
	HSPQ Factor B	0.11	
	HSPQ Factor Q4	0.08	
	HSPQ Factor I	-0.14	
	HSPQ Factor G	0.05	
	HSPQ Factor A	0.06	
	HSPQ Factor H	-0.06	
	HSPQ Factor F	-0.04	
	HSPQ Factor Q2	-0.03	
	HSPQ Factor E	-0.02	0.36
All Predictors	HSPQ Factor J	0.25	
	HSPQ Factor C	-0.21	
	HSPQ Factor D	0.21	
	HSPQ Factor Q3	0.16	
	HSPQ Factor B	-0.08	
	HSPQ Factor Q4	0.05	
	HSPQ Factor I	0.01	
	HSPQ Factor G	-0.02	
	HSPQ Factor A	0.05	
	HSPQ Factor H	-0.13	
	HSPQ Factor F	0.17	
	HSPQ Factor Q2	-0.04	
	HSPQ Factor E	-0.06	
	PAT Mathematics	0.42	
	HSPQ Factor O	-0.00	
	Mean Norm. Arithmetic Treatment	0.22 -0.18	
	PAT Vocabulary	0.05	0.63

Note: HSPQ Factor C was forced out of the equation by PAT Mathematics, and only came in again at the 15th step.

Table 17: Regression Analyses: Regression Coefficients and their Standard Errors, by Predictor Variable

	Algorithmic Skill as Dependent Variable		Coding Knowledge as Dependent Variable	
	B	Std. Errors of B[1]	B	Std. Errors of B[1]
PAT Mathematics	0.09	0.06;0.06	0.18	0.07;0.08
PAT Vocabulary	0.10	0.04;0.04	0.017	0.05;0.06
Arithmetic	0.08	0.33;0.33	0.68	0.44;0.46
HSPQ Factor A	-0.03	0.11;0.11	0.07	0.15;0.15
HSPQ Factor B	-0.04	0.26;0.26	-0.22	0.36;0.36
HSPQ Factor C	-0.07	0.13;0.13	-0.28	0.18;0.18
HSPQ Factor D	-0.10	0.11;0.11	0.25	0.15;0.15
HSPQ Factor E	-0.19	0.13;0.14	-0.08	0.19;0.19
HSPQ Factor F	-0.08	0.12;0.13	0.18	0.17;0.17
HSPQ Factor G	0.003	0.11;0.11	-0.02	0.15;0.16
HSPQ Factor H	-0.02	0.13;0.13	-0.14	0.18;0.18
HSPQ Factor I	-0.06	0.09;0.09	0.01	0.12;0.12
HSPQ Factor J	0.23	0.12;0.12	0.30	0.16;0.16
HSPQ Factor O	-0.06	0.13;0.13	-0.002	;0.18
HSPQ Factor Q2	-0.06	0.13;0.13	-0.05	0.17;0.17
HSPQ Factor Q3	0.04	0.12;0.12	0.20	0.16;0.16
HSPQ Factor Q4	-0.10	0.14;0.14	0.07	0.18;0.19
Treatment	0.55	0.37;0.37	-0.74	0.50;0.51

[1] In each case, the first-named standard error is that obtained when the academic predictors were forced into the regression equation first (as in Tables 13, 15). The second-named standard error is that obtained when the personality variables were forced in first (as in Tables 14, 16).

Though analysis of the regression coefficients cannot be rigorously applied to this situation, some observations about them may be useful in pointing the way to later research. Of the HSPQ factor scores, only one had a regression coefficient that was more than 1.6 times its standard error in any of the regression equations. The factor that had this property - in every equation, in fact - was Factor J. How the HSPQ Manual describes this and other factors is shown in Table 18 below.

Table 18: Brief Description Of The Fourteen HSPQ Personality Factors, by factor.

LOW STEN SCORE DESCRIPTION (1-3) A boy or girl with low score is:	ALPHABETIC DESIGNATION OF FACTOR	HIGH STEN SCORE DESCRIPTION (8-10) A boy or girl with high score is:
Reserved, Detached, Critical, Aloof, Stiff	A	Warmhearted, Outgoing, Easygoing, Participating
Less Intelligent, Concrete-Thinking, Of Lower Scholastic Mental Capacity	B	More Intelligent, Abstract-Thinking, Bright, Of Higher Scholastic Mental Capacity
Affected by Feelings, Emotionally Less Stable, Easily Upset, Changeable, of Lower Ego Strength	C	Emotionally Stable, Mature, Faces Reality, Calm, of Higher Ego Strength (not the same as "egotistical")
Undemonstrative, Deliberate, Inactive, Stodgy, Phlegmatic	D	Excitable, Impatient, Demanding, Overactive, Unrestrained
Obedient, Mild, Easily Led, Accommodating, Submissive	E	Assertive, Competitive, Aggressive, Stubborn
Sober, Taciturn, Serious	F	Enthusiastic, Heedless, Happy-Go-Lucky
Disregards Rules, Expedient, Has Weaker Superego Strength	G	Conscientious, Persistent, Moralistic, Staid, Has Stronger Superego Strength
Shy, Timid, Threat-Sensitive	H	Adventurous, "Thick-Skinned", Socially Bold
Tough-Minded, Rejects Illusions	I	Tender-Minded, Sensitive, Clinging, Over-Protected
Restful, Likes Group Action	J	Circumspect, Individualism, Reflective, Internally Restrained
Self-Assured, Placid, Secure, Complacent, Untroubled	O	Apprehensive, Self-Reproaching, Insecure, Worrying, Guilt Prone
Sociably Group-Dependent, A "Joiner" and Sound Follower	Q_2	Self-Sufficient, Prefers Own Decisions, Resourceful
Uncontrolled, Lax, Follows Own Urges, Careless of Social Rules, Has Low Integration	Q_3	Controlled, Socially-Precise, Self-Disciplined, Compulsive, Has High Self-Concept Control
Relaxed, Tranquil, Torpid, Unfrustrated, Composed	Q_4	Tense, Driven, Overwrought, Frustrated, Fretful

Source: Cattell and Cattell (1968b, p.12)

Other factors showing somewhat weaker positive regression coefficients in both sets of equations are factors B and Q3, while coefficients of similar magnitude but negative sign in both sets are those for Factors C and I.

Of particular interest is the factor D. With all variables in the regression equation on Coding Knowledge score, its regression coefficient (B) was +0.25 with a standard error of 0.15; when Algorithmic Skill was the dependent variable, it had B = -0.10, with 0.11 for the standard error.

Comparison of Teaching Strategies

The differential effects of the two teaching strategies (flowcharting-then-coding, coding-then-flowcharting) were tested by an analysis of variance procedure. The same procedure was also used to yield additional information about whether teaching actually makes any significant difference to the level of achievement, or whether maturation alone would so as much. A one-way ANOVA was performed on three groups: one group taught by each of the two methods mentioned above, and a third control group that had had no instruction in programming at all. As can be seen in Table 19, the value of F obtained for comparison among these three groups was significantly different from zero, at the 0.05 level. Since the difference between the two taught groups was clearly not significant (see Table 20), Dunnett's (1955) method was employed to compare the two taught groups with the control. The values of Dunnett's t statistic, also shown in Table 20, were both significant under a two-tail test at the 0.05 level.

Table 19: Algorithmic Skill Test Scores, 1974 Sample, Summary of Analysis of Variance.

Source of Variation	Degrees of Freedom	Sum of Squares	Mean Squares	F Ratio
Between groups	2	315.5	157.7	16.9 (p < 0.001)
Within groups	96	896.6	9.3	
Total	98	1212.1		

Table 20: Algorithmic Skill Test Scores, 1974 Sample, Differences Between Taught Groups and Control.

Group	Number of subjects	Group Mean	Dunnett's t
Flowcharting -then- Coding	36	11.3	5.0 (p <0.01)
No Programming (Control)	32	7.6	5.1 (p <0.01)
Coding -then- Flowcharting	31	11.5	

The Coding Knowledge Test was administered only to the two taught groups, as the form of the test rendered it incomprehensible to anyone who had not undertaken a course in CESIL. A t-ratio was therefore used to compare these two groups, with the result shown in Table 21.

Table 21: Coding Knowledge Test Scores, 1974 Sample, Differences between Taught Groups.

Group	Flowcharting then Coding	Coding then Flowcharting
Group mean:	9.0	6.8
Standard Deviation:	3.7	4.2
Number of Subjects:	36	30

Student's t: 2.2 ; 64 degrees of freedom ($p < 0.05$)

This value was significant under a two-tail test at the 0.05 level, and the hypothesis of no difference was therefore rejected.

In sum, it may be said that those subjects who received instruction did perform significantly better on the Algorithmic Skill Test than those who did not, but that whether the instruction in flowcharting preceded or followed instruction in coding made no significant difference. On the other hand, end-of-course scores on the Coding Knowledge Test were significantly higher under the flowcharting-then-coding strategy than under the coding-then-flowcharting strategy.

CHAPTER VI

CONCLUSIONS

Summary of the Results

The most important individual statistics to emerge from this study were the first-order partial correlation coefficients, factored by Otis IQ, of Algorithmic Skill vs Coding Knowledge (0.46), Algorithmic Skill vs Composite Mathematics Attainment Score (0.48) and Coding Knowledge vs Composite Mathematics Attainment Score (0.60). Among the subscores of this set, the most revealing relationships were the weak correlations between CESIL syntax and Arithmetic, and between Algorithmic Skill and Logic.

Attempts to predict success in the Computer Studies 1 course by a regression model showed that past academic attainments were the most powerful predictors of both Algorithmic Skill and Coding Knowledge at the end of the course. In general, results of other studies with adults were confirmed with the younger subjects studied in the present investigation; however, some suggestive differences were also found.

Coding Knowledge, as measured at the end of the programming course, was shown to be significantly higher if instruction in coding followed instruction in flowcharting rather than vice versa. For Algorithmic Skill, no corresponding difference was found, but it was shown that pupils who had been instructed in programming did significantly better on the test than those who had not.

Interpretation of the Results

The evidence obtained in this investigation was considered to support the view that Algorithmic Skill and Coding Knowledge are two distinct abilities, exhibiting features that differ in important ways.

Beliefs about the statistical independence of COBOL Coding Knowledge and Algorithmic Skill were nevertheless shown to be untenable. The two tests used employed tasks which could hardly have been made more dissimilar without losing content validity, but the zero-order correlation coefficient obtained was high enough to justify rejecting the null hypothesis of no correlation. Nor could the effect be explained away as a phenomenon of general intelligence or testwiseness, as was shown when the first-order coefficient factored by Otis IQ was also found to be high enough to justify rejecting the null hypothesis.

This evidence strongly suggests that a sizeable general programming ability exists, pervading both tests, and that this general programming ability may differ in important ways from general intelligence. There was also very considerable correlation with mathematical attainment, which again was shown to be due not only to general intelligence but also to other mutually contributing abilities. It should be remembered, however, that the Computer Studies 1 course is concerned entirely with numeric processing, and that courses and languages for non-numeric processing may reveal significantly different relationships with mathematical attainment.

Conclusions drawn from the more detailed correlational survey of the relationship between mathematical attainment and programming attainments must necessarily be treated with caution, because of the lower reliabilities of the subscores. However, the two lowest partial correlation coefficients (factored by Otis IQ) merit some comment. The first of these, Arithmetic vs COBOL Syntax, results from a low zero-order coefficient between the two variables, and comes as no surprise since the two kinds of task are very dissimilar. The second, that between Algorithmic Skill and Logic, may be explained partly in terms of the relatively high correlation of each with Otis IQ, but also suggests that there is some conceptual difference between the dynamic logic of flowcharting (concerned with procedures) and the propositional logic of mathematics (concerned with relationships). Again, part of the explanation may lie in the language-related

nature of the questions in the Logic subtest, and the comparatively nonverbal character of the tasks in the Algorithmic Skills Test.

The regression study clearly demonstrated that for this sample, academic attainments were the most efficient predictors of success in the programming course. This finding accords with that of C.A. Alspaugh, who, working with college undergraduates, concluded (1972) that

> "...the mathematical background of a computer programming student appeared to be the major influencing component for achievement in BAL ..." (p.97)

PAT Mathematics, PAT Vocabulary and ACER Arithmetic predicted 51% of the variance in Algorithmic Skill Test score. HSPQ scores, in adding only another 14% would, in any practical situation, provide so little extra precision that the effort of gathering them would be unjustified. Where Coding Knowledge was the dependent variable, however, personality variables, though still the weaker predictors, contributed relatively more: the percentages corresponding to the two above were 43% and 20%. (In fact, an intermediate calculation in the stepwise regression procedure revealed that the three academic predictors together with HSPQ factors D,H,J and C predicted 59% of the variance in Coding Knowledge score.)

Turning to the individual personality predictors used in the present study, the strength of HSPQ Factor J (Circumspect individualism, Reflective, Internally restrained) confirms both Alspaugh's finding with older students and the popular stereotype of the computer programmer. That Factor B, which is a general intelligence score, should be weighted heavily is no surprise, and neither are the indications from Factors Q3 and I that the high achiever tends to be "controlled","socially precise", "tough-minded". It is surprising, though, to find the negative weighting for Factor C, indicating that the superior programmer is "affected by feelings,

emotionally less stable, easily upset, changeable, of lower ego strength". Such a result conflicts with the findings of Alspaugh and Morris and Wise (see above, p. 14) for older students, and if truly significant, would indicate a change in the nature of the programming task (or possibly the learning processes involved) with age. It is not easy to see what such a change would be.

Also difficult to explain is the variation in Factor D, from a negative weighting ("deliberate, stodgy, phlegmatic") with Algorithmic Skill to a positive one ("excitable, impatient, demanding, overactive, unrestrained") with Coding Knowledge. Both this result and the previous one merit further investigation.

The study comparing the effects of different teaching sequences demonstrated that even when the material presented is the same, the lessons used are the same, the school environment is the same and the teaching style is the same, variations in the sequence of lessons can produce significant differences in the Coding Knowledge scores obtained by students at the end of the course. Since both teaching plans used in this experiment provided for a month of practical work between the conclusion of instruction and the beginning of testing, the difference cannot merely have been the result of short-term memory, and must therefore be regarded as educationally as well as statistically significant.

On the other hand, the same effect was not found with Algorithmic Skill. To the contrary: the group means for the groups taught in the two different sequences showed a close correspondence. Had the study demonstrated no more than that, the possibility would still have remained that in the matter of Algorithmic Skill, both strategies had failed equally to produce any increases in score that would not have resulted from maturation or other factors beyond the programming course entirely. But this possibility was eliminated when the differences in performance between the taught groups and the untaught control were found to be significant.

To sum up, evidence was found that teaching for Algorithmic Skill does improve scores, but the timing of the instruction may not be important; for Coding Knowledge, timing does matter. If both Algorithmic Skill and Coding Knowledge are seen to be important outcomes of a programming course, it would appear that better all-round results are to be obtained by teaching flowcharting before coding than by using the opposite sequence.

The tests used in this investigation may be considered to be outcomes of the project, since they are applicable to a range of situations beyond the present one. Further development of the Coding Knowledge Test for CESIL is probably unjustified, as its reliability is high and could even be increased by using both forms A and B of the test. And of course its usefulness is limited to that minority of situations in which CESIL is employed as the programming language.

But the Algorithmic Skills Test is applicable to all programming environments in which flowcharts are used - at this stage the great majority. Although the test is fairly reliable, a satisfactory rationale for constructing such tests is still lacking, and therefore construct validity cannot be established. Whether such a rationale will ever be found is a matter for conjecture, but it is believed that the face validity of the test is high, and furthermore it can be seen to be almost language-independent - whether one is thinking of programming languages or of natural language. Therefore further development of this test is justified, in particular to refine the process of measuring a subject's ability to perceive and manipulate algorithms at a global level.

Suggestions for Further Research

This study has examined phenomena associated with a particular age-group of students learning how to program in a particular language. Immediate, and important, questions arise about the effects of changing the language and using older or even younger learners.

Dealing first with the question of studying different languages, some serious difficulties present themselves: for example, how to construct a test of coding knowledge for a high-level language with a wide range of acceptable syntactical forms like COBOL or a structured language like Algol? The most effective way of dealing with these major difficulties might be a "bootstrapping" approach, constructing and proving coding knowledge tests for progressively augmented subsets of, say, Fortran. Careful analysis of the diagnostic routines incorporated in the compiler would be necessary, but perhaps not sufficient, to develop achievement tests of high content validity.

Not only would there be problems in constructing a test of coding knowledge; measuring algorithmic skill in other environments would present difficulties that were circumvented in the present investigation. In the present state of the programming art, the language in which an algorithm is ultimately to be coded does exert subtle influences on the kind of algorithm the programmer develops. Sometimes the influence is less subtle: Algol programs are often difficult to envisage in flowchart form, and an alternative kind of pictorial representation - the structure diagram - has been developed to be used with Algol and similar structured languages. Is there one kind of algorithmic skill relating to flowcharts, and another for structure diagrams? This researcher is inclined to believe so. It could be that the proper course is to develop an omnibus Algorithmic Skills Test containing two sections, one on sequenced algorithms and one on structured algorithms. Certainly both kinds of skill are needed by a COBOL programmer, who may employ

both structured and unstructured strategies in a single program.

The outcome of all this effort in test construction would be a generalized construct or linked set of constructs defining algorithmic skill, and a whole range of coding knowledge constructs, one for each major programming language. The researcher suggests that if instruments to measure these constructs were available, the empirically-determined correlation between algorithmic skill and the appropriate kind of coding knowledge would be a feature of a programming language useful in comparing it with other languages. This is not to say that a high correlation between coding knowledge and algorithmic skill would necessarily be regarded as highly desirable. In the academic environment, where the person who designs an algorithm also codes it, and where unsuccessful programmers have the option of doing other things, such a high correlation may well be desirable; in the commercial environment, where programming is often a team effort, a lower correlation may be advantageous. Be that as it may, studies of the kind proposed would permit the very considerable theory of programming languages to be linked with practical psychology.

In a similar way, studies relating programming skills to attainments in other areas of study may reveal useful relationships between particular programming languages (or courses) and particular fields of activity. The study of CESIL programming and mathematics in this investigation was such a study, and the results obtained from it lead the researcher to conject that considerable transfer of learning might be found between the Computer Studies 1 course and a standard New Zealand mathematics course. In a utopian vision, he sees programming languages being subjected to "aptitude tests" to determine their suitability for use in various areas of application.

The other major variable requiring further study is that of age. The present investigation has shown that the psychology of computer programming for twelve-year-olds is not identical with that for college students. It is suggested that replications of the present study at different levels in the secondary school system would provide valuable data on variations in the learning processes with intellectual and emotional maturation. For example, the researcher hypothesises that seventh-form students learn COSIL more slowly than second-formers; if true, this would be useful information for curriculum designers in deciding the proper place for computer programming in the school curriculum.

Summary

In this investigation, the Computer Studies 1 course was taught to representative groups of Form 2 children in New Zealand Intermediate schools. At the end of each course of instruction, achievement on Algorithmic Skill Test and the Coding Knowledge Test was measured.

Scores on these two tests were considered to measure two distinct abilities, and considerable evidence for this view was found. It was also found that the two scores were moderately correlated, and that each correlated moderately with the Composite Mathematics Attainment Score. Factoring the correlations by Otis IQ showed that they were not due to general intelligence alone, and a matrix of correlations between subscores on the various tests permitted examination of the fine structure of the relationships involved.

It was further shown that taken together, scores from the PAT Mathematics and Vocabulary Tests and the ACER Arithmetic Tests predicted attainment in the programming course more effectively than the fourteen factor scores of the High School Personality Questionnaire. Some similarities and some differences

were discovered between the results of this prediction study and similar studies with adults.

Finally, the effects of teaching flowcharting before coding were compared with those of teaching coding before flowcharting, and of teaching no programming whatever. It was established that algorithmic skill was improved by instruction, but neither teaching strategy was found to be superior to the other in this respect. On the other hand, Coding Knowledge Test scores were found to be significantly higher in the group which had been taught flowcharting before coding.

Appendix A: Outlines of the Courses Used for the 1974 Comparison of Instructional Strategies

The individual lessons used in the teaching programmes are listed below, together with the textbook material on which each lesson was based. In the description, the abbreviation CES is used for Computer Studies 1 (ICL/CES (1971)), SOM for The Shape of Mathematics - 2A (Reed Mathematics Group (1970)), and UF for Units in Mathematics. T7: Formulae (Reed Mathematics Group (1973)).

Lessons about computer systems in general

G1: Parts of a computer system (CES, 3 - 4)
G2: How the parts of a computer system combine (CES, 5 - 8)
G3: Computer Hardware (CES, Chapter 4)

Lessons about flowcharting

A1: Basic flowchart concept (CES, 79 - 81)
A2: Flowcharting a journey (CES, 83 - 86)
A3: Branching decisions (CES, 87 - 91)
A4: Constructing flowcharts (SOM, 82 - 84)
A5: Iterative processes (SOM, 84 - 86)
A6: Mathematical formulae (UF, 1 - 7)
A7: Harder formulae (UF, 9 - 16; CES, 95 - 100)

Lessons about coding

C1: Notion of storage; IN (CES, 11 - 15)
C2; STORE (CES, 15 - 17)
C3: LOAD, OUT, HALT, in the context of a class-generated program for printing a telephone directory.
C4: Coding conventions (CES, 17 - 23). Sample program from CES P. 28 prepared for processing.
C5: Discussion of output resulting from sample program. Arithmetic in CESIL (CES, 27)
C6: Further arithmetic in CESIL (CES, 35 - 36)

C7: Programs for problems (CES, 37 - 42). Sample program from CES pp. 35 - 6 prepared for processing.

C8: Branching instructions (CES, chapter 7)

Free-play experiential sessions

E1, E2, E3, E4: Pupils design and code their own programs to be batched, run on the computer and returned before the next session.

Each lesson occupied one school period of 40 minutes. They were given once a week, normally on Wednesday morning. The order in which the lessons were presented to the three groups was as follows:

Group AC:

G1,G2,A1,G3,A3,A2,A4,A5,A6,A7,C1,C2,C3,C4,C5,C6,C7, E1,E2,E3,E4.

Group CA

G1,G2,C1,G3,C2,C3,C4,C5,C6,C7,A1,A2,A3,A4,A5,A6,A7, E1,E2,E3,E4.

Group Control

A1.

(It was decided that total unfamiliarity with flowcharts prior to testing might cause "symbol shock" in the control group and so invalidate the comparison between this untaught group and the two taught groups. Lesson A1 was therefore presented to Group Control, one week before testing.)

Appendix B:

Project PEACE

Standardization of the Test

MATHEMATICAL ATTAINMENT FORM II

ADMINISTRATION

Children require pencils, erasers and answer sheets for recording answers to multiple-choice questions. Seating should be arranged so that co-operative work is impossible. Conditions must be such as to allow every child to complete the test without interruption, i.e. for about 45 minutes.

Test booklets are distributed. The teacher says: Keep your booklet closed. Write your name on the answer sheet. Also write your group number and personnel number.

The teacher then explains the method of recording answers to multiple-choice questions on the answer sheet. (The answer sheets used in the PEACE project have special dummy "S" boxes which may be used for this purpose.)

After dealing with questions, the teacher says: You have about half an hour to do this test, but there's no actual time limit so you don't have to rush. I want to be sure that everybody does every question. Now is there anything at all you want to know about the test before we start? Ask now, because it's your last chance.

After dealing with questions:
Open your booklet and start the test.

If a pupil wants help during the test, the teacher may assist by repeating the substance of the instructions given at the beginning. No other help can be given.

As children complete the test, they should be encouraged to spend some time in checking, particularly that they

have answered every question. This done, their answer sheets and test booklets should be collected as they finish. The teacher should check each answer sheet to see that every question has been answered; if not, booklet and answer sheet should be returned to the pupil for completion.

SCORING

For every item, one point is scored if the correct answer is selected, none for incorrect or multiple answers. If the test has been correctly administered, there should be no cases of missing or invalid responses.

Items are grouped according to content:

Numerical Relationships	Items 1 - 10
Logical Relationships	Items 11 - 20
Spatial Relationships	Items 21 - 30

and separate subscores for these categories may be extracted.

A computer program written in B6700 Fortran is available for scoring this test.

FORM 76 B

MASSEY UNIVERSITY

Project PEACE

Mathematical Attainment

FORM II

Do not write on this booklet

Use your answer sheet to record your answers.

DO NOT OPEN UNTIL YOU ARE TOLD

1. Only one of these statements is true
   (A) There is a last whole number
   (B) There is no first whole number
   (C) When one whole number is added to another, the answer *is not necessarily* a whole number
   (D) When one whole number is subtracted from another, the answer *is not necessarily* a whole number

2. $(28 - 20) - (7 + 1)$ equals
   (A) 14 (B) 0 (C) 16 (D) $^-2$ (E) 2

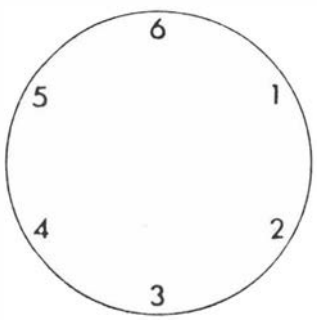
3. $^-2 + {}^+5$ equals
   (A) 7 (B) 3 (C) $^-3$ (D) $^-7$

Questions 4, 5:
Here is a string of numbers: $\frac{1}{2}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{16}$ . . .

4. The *next* number in the sequence would be
   (A) $\frac{1}{18}$ (B) $\frac{1}{32}$ (C) $\frac{1}{20}$ (D) 16 (E) 32

5. The *seventh* number in the sequence would be
   (A) $\frac{1}{16}$ (B) $\frac{1}{32}$ (C) $\frac{1}{64}$ (D) $\frac{1}{128}$ (E) $\frac{1}{256}$

Use this 'clock' for questions 6, 7.



In this type of "clock" arithmetic,

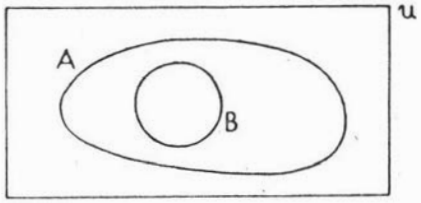
6. $1 - 4 =$ (A) $^-3$ (B) 0 (C) 1 (D) 3 (E) 5
7. $3 \times 4 =$ (A) 4 (B) 12 (C) 6 (D) 3 (E) 0

8. A simpler numeral for $\frac{a^5 b}{a^2 b^2}$ is
   (A) a^3 (B) a (C) ab (D) $\frac{a}{b}$ (E) $\frac{a^3}{b}$

9. A simpler numeral for $\frac{16 + 8}{32 + 12}$ is
   (A) $\frac{17}{16}$ (B) $\frac{5}{6}$ (C) $\frac{1}{2}$ (D) $\frac{5}{7}$ (E) $\frac{6}{11}$

10. $\frac{3}{5} \div \frac{a}{b}$ is equivalent to
    (A) $\frac{3a}{5b}$ (B) $\frac{5b}{3a}$ (C) $\frac{3b}{5a}$ (D) $\frac{5a}{3b}$

11. In this Venn Diagram we see that



(A) $A \subset B$ (B) $A \supset B$ (C) $A \cap B$ (D) $A \cup B$ (E) $A = B$

12. In Question 11

(A) $A \cap B = A$ (B) $A \cap B = B$ (C) $A \cap B = \mathcal{U}$ (D) $A \cap B = \phi$

(E) $A \cap B = B'$

Questions 13-20 refer to these sets:

$\mathcal{U}$ = { 4, 5, 6 . . . 20 }

P = { 6, 8, 10 . . . 20 }

Q = { 5, 7, 9 . . . 19 }

R = { 6, 9, 12, 15, 18 }

13. Which of the following statements is true?

(A) $P = Q$ (B) $P > Q$ (C) $P \supset Q$ (D) $R \subset Q$ (E) $P \subset \mathcal{U}$

14. $P \cap Q$ is the set

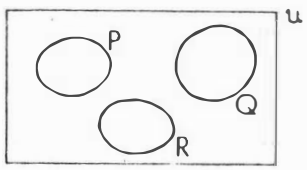
(A) ϕ (B) R (C) {5, 6, 7, 8 . . . 20} (D) {6, 12, 18} (E) $\mathcal{U}$

15. $P \cap R$ is the set

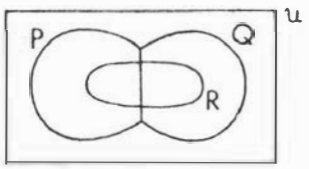
(A) ϕ (B) R (C) {5, 6, 7, 8 . . . 20} (D) {6, 12, 18} (E) $\mathcal{U}$

16. The Venn diagram for these sets is

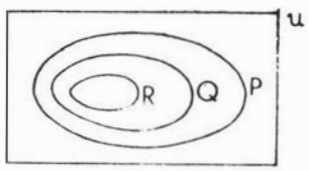
(A)



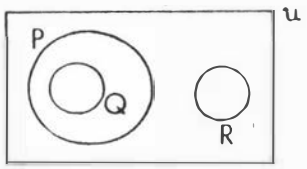
(B)



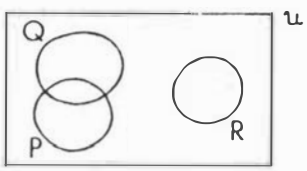
(C)



(D)



(E)



17. $P \cup Q$ is the set

(A) ϕ (B) R (C) {5, 6, 7, 8 . . . 20} (D) {6, 12, 18} (E) $\mathcal{U}$

18. P' is the set

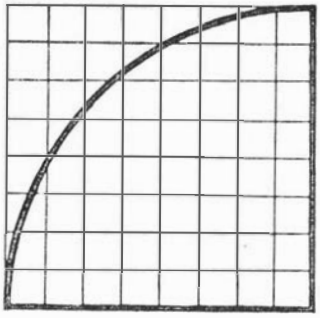
(A) $\mathcal{U}$ (B) P (C) $Q \cup \{4\}$ (D) R (E) {4, 6, 8 . . . 20}

19. $P \cap P$ is the set

(A) ϕ (B) $\mathcal{U}$ (C) P (D) P' (E) $Q \cup \{4\}$

20. $(P \cup Q) \cap R$ is the set

(A) ϕ (B) {6, 12, 18} (C) R (D) {6, 7, 8 . . . 20} (E) $\mathcal{U}$

21. 

The area of this shape is about

(A) 30 square units (B) 44 square units

(C) 50 square units (D) 56 square units

(E) 64 square units

22. Which of these shapes has the greatest volume?

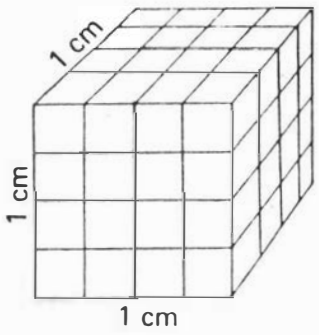


Figure 1

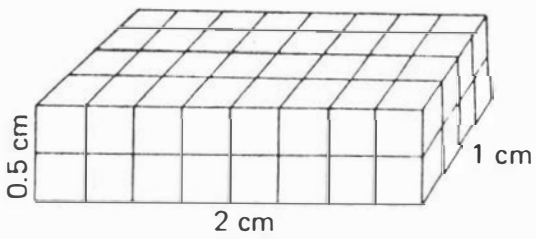


Figure 2

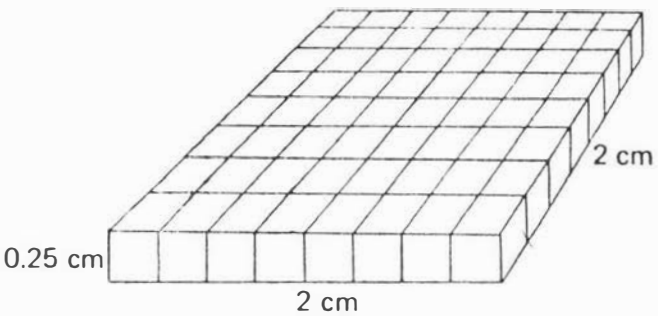


Figure 3

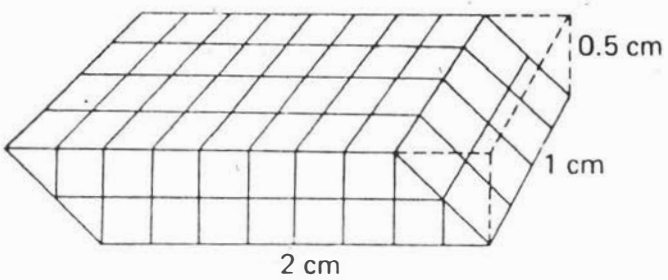
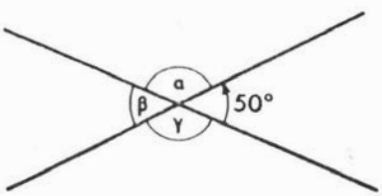


Figure 4

(A) Figure 1 (B) Figure 2 (C) Figure 3 (D) Figure 4

(E) None of them—they all have the same volume

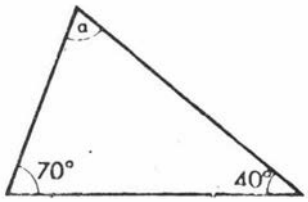
23.



In this diagram, the size of angle α must be

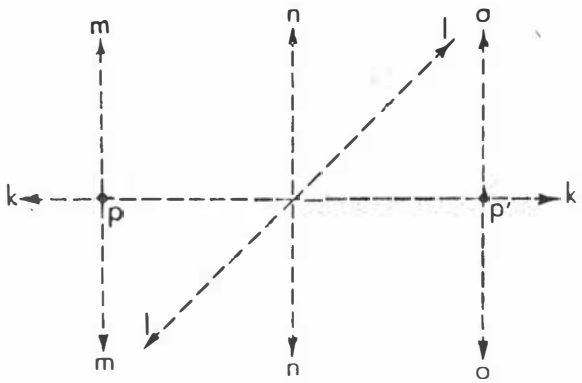
(A) 50° (B) 130° (C) 150° (D) 230° (E) 310°

24. In this diagram:



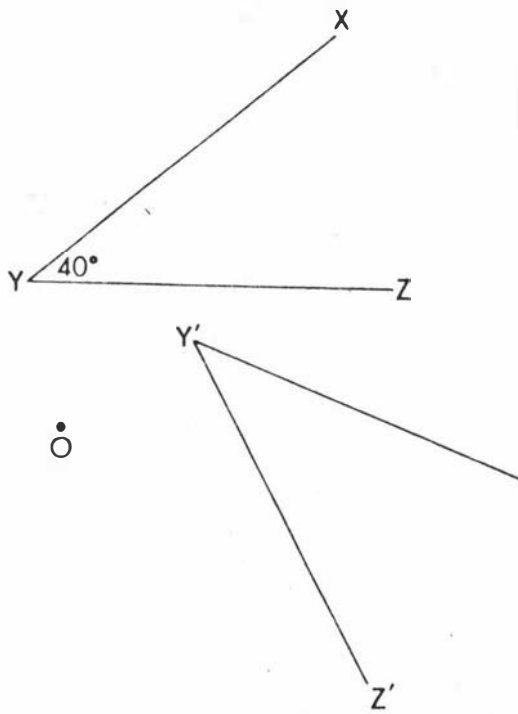
the size of angle α must be
(A) 40° (B) 70° (C) 90° (D) 0° (E) 30°

25.

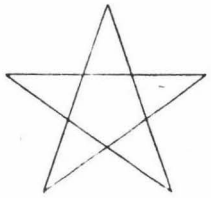


To map P onto P' by reflection, you would use
(A) mirror line k (B) mirror line l (C) mirror line m (D) mirror line n
(E) mirror line o

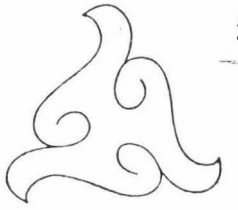
26. In the diagram, figure XYZ has been rotated 50° about O to X'Y'Z'. Angle XYZ = 40°



The size of angle X'Y'Z' is
(A) 40° (B) 50° (C) 10° (D) 90°



27. How many axes of symmetry does this figure have?
(A) None (B) 1 (C) 5 (D) 10 (E) an indefinite number



28. How many axes of symmetry does this figure have?
(A) 0 (B) 1 (C) 3 (D) 6 (E) an indefinite number

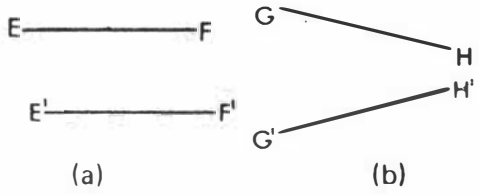


Figure 2

29. In Figure 2 (a), you could map EF onto E' F' by
(A) rotation, but not by reflection
(B) reflection, but not by rotation
(C) either reflection or rotation
(D) neither reflection nor rotation

30. In Figure 2 (b), you could map GH onto G' H'
(A) rotation, but not by reflection
(B) reflection, but not by rotation
(C) either reflection or rotation
(D) neither reflection nor rotation

APPENDIX C

Project PEACE

Standardisation of the Test

PROGRAMMING KNOWLEDGE Mark 2

ADMINISTRATION

Children require pencils and erasers. They must be placed so that co-operative work is impossible. Conditions must be such as to allow every child to complete the test without interruption, i.e. for 45 minutes.

Test booklets are distributed. The teachers says:

Keep your booklet closed. Write your name on the front. Also write your group number and personnel number. When everyone has done that, we'll open the booklets.

This done, the teacher says:

Now open your booklet and look at page 1. You will see some CESIL programs written on coding sheets. Let's look at the top one. The program itself is written in columns 1, 2 and 3 as usual, and on the effects side there are columns for store locations called A, B and C. There is also a column for writing down the output.

The teacher then makes a copy of the program on the blackboard and works the EFFECTS section through, producing a result as shown in the lower part of p.1, then says:

Did you see where the output goes? We didn't actually need the store locations A, B and C that time did we? Some of the other programs will, though. Is there anything you want to ask now?

After dealing with questions, he says:

Now turn to page 2, and you'll find one that you have to do. Try it now.

After the children have attempted this in silence,

How did you get on with that one? Did you get the same as me?

The Teacher works example on blackboard. After dealing with question:

You have about half an hour to do this test, but there's no actual time limit so you don't have to rush. I want to make sure that everybody does every question.

Before we start, there are some other things I want to show you. Turn to page 11. On this page there are two programs to work out, so make sure you do them both. There are one or two other places where that happens, so

watch out for them. The other thing I want to show you is on page 12. There are some questions at the end like this. There's a group of four CESIL commands, but only one of them is written correctly.

Teacher discusses criteria for selecting correct answer in sample, then says:

Now is there anything at all you want to know about the test before we start? Ask now, because it's your last chance.

After dealing with questions:

Then turn to page 3 and start the test.

If a pupil wants help during the test, the teacher may assist by repeating the substance of the instructions given at the beginning. No other help can be given (this applies particularly to p.7!)

As children complete the test, they should be encouraged to spend some time in checking, particularly that they have answered every question. This done, their booklets should be collected as they finish.

SCORING

For every program in pp 3-11 there is a row or column of key entries which the scorer notes. (Entries in other rows or columns are ignored.) These key entries are marked in red in the scoring key.

A given program scores 1 if its key entries correspond to those of the scoring key in these respects:

(1) The correct numbers are shown

(2) They are in the correct order

NOTES:

It is common for a child to write a number one space too high or too low in the output column. If this does not affect the order of the key entries, it is not penalised.

Positive numbers may be written with or without the '+' sign. Thus 5 and +5 are equivalent. 5 and −5 are not, however. Any of 0, +0, −0 are treated as equivalent symbols for zero.

For pp. 5 and 11, the page score may be 0, 1 or 2.

For p.13, each correctly placed tick scores 1 provided only one option is ticked. Incorrect or multiple choices score 0. The page score will lie between 0 and 6.

MASSEY UNIVERSITY

Project PEACE

Programming Skills Test FORM A

CODING KNOWLEDGE

(Formerly called
Programming Knowledge)

Mark 2c

Write your name here: _____

Group Number: _____

Personnel Number: _____

This is a test to see how much you know about the computer language CESIL.

First of all there are 14 programs which have been written out for you, like this:

EFFECT

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C		ØUT PUT
	▽	IN	▽							
	▽	ØUT	▽							
	▽	ØUT	▽							
	▽	HALT	▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							

Data	+5	+7				

You have to fill in the EFFECT side, like this:

EFFECT

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C		ØUT PUT
	▽	IN	▽		5					
	▽	ØUT	▽		5					5
	▽	ØUT	▽		5					5
	▽	HALT	▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							

Data	+5	+7				

Did you see where the output goes?

TURN OVER

Now here is one for you to try:

EFFECT

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C		OUT PUT
	▽	IN	▽							
	▽	IN	▽							
	▽	IN	▽							
	▽	ØUT	▽							
	▽	HALT	▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							

Data	+ 5	+9	+12	+7		

ASK YOUR TEACHER NOW if there is anything you do not understand. You cannot ask once the test has started.

When you are told, turn this page and start the test.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO START

PROGRAM STØRE

EFFECT

(1) (2) (3)

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUT PUT
	IN								
	STØRE	A							
	IN								
	STØRE	B							
	STØRE	C							
	STØRE	A							
	HALT								
									(4-13)

Data	+5	+9					

Function of Program

PROGRAM LØAD

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STØRE	A							
	IN								
	ØUT								
	LØAD	A							
	ØUT								
	HALT								
									(14-17)

Data	+5	+9					

Function of Program

CESIL CODING SHEET 4/361(8.70)

PROGRAM ADD

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					OUTPUT
	IN								
	STORE	A							
	IN								
	ADD	A							
	OUT								
	HALT								
									(18-14)

Data	+4	+1					

THERE ARE TWO QUESTIONS ON THIS PAGE !!!

PROGRAM MULTIPLY

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					OUTPUT
	IN								
	STORE	A							
	IN								
	MULTIPLY	A							
	OUT								
	HALT								
									(20-22)

Data	+8	+3					

PROGRAM SUBTRACT

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					OUT PUT
	IN								
	STORE	A							
	IN								
	SUBTRACT	A							
	OUT								
	HALT								
									(23-25)

Data	+4	+12					

Function of Program

CESIL CODING SHEET 4/361(8.70)

PROGRAM DIVIDE

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					OUTPUT
	IN								
	STORE	A							
	IN								
	DIVIDE	A							
	OUT								
	HALT								
									(26-28)

Data	+13	+3					

Function of Program

CESIL CODING SHEET 4/361(8.70)

PROGRAM LØADCØNST EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			OUTPUT
	IN								
	LØAD	+5							
	ØUT								
	HALT								
									(29-31)

Data	+3						

Function of Program

CESIL CODING SHEET 4/361(8.70)

PROGRAM CØNSTS

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			OUTPUT
	IN								
	STØRE	B							
	ADD	+12							
	ØUT								
	IN								
	SUBTRACT	+7							
	ØUT								
	IN								
	MULTIPLY	+5							
	ØUT								
	HALT								
									(32-42)

Data							
	+9	+7	+2	+17			

Function of Program

PROGRAM J U M P

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUT PUT
	IN								
	JUMP	B3							
	IN								
B3	ØUT								
	HALT								
									(43-45)

Data	+5	+9					

Function of Program

CESIL CODING SHEET 4/361(8.70)

ODING SHEET

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C			ØUT PUT
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
B4	▽	HALT	▽								
	▽		▽								(46-56)

Data							
	+5	+9	−4	+0	+3	+4	

PROGRAM JINEG EFFECT

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C			ØUT PUT
	▽	IN	▽								
	▽	JINEG	▽	B6							
	▽	IN	▽								
	▽	JINEG	▽	B6							
	▽	IN	▽								
	▽	JINEG	▽	B6							
B6	▽	ØUT	▽								
	▽	HALT	▽								
	▽		▽								(57-59)

Data							
	+3	−2	+7				

On the next page you will find six groups of four CESIL instructions, like this:

	COLUMN 2		COLUMN 3		
▽	ØUT	▽	PQ		1
▽	PQ	▽	ØUT		2
▽	ØUT	▽			3
▽	ØUT	▽	+5		4

In each group, only <u>one</u> instruction is written correctly. Put a tick beside the one that is <u>correct</u>, like this:

▽	ØUT	▽	PQ		1
▽	PQ	▽	ØUT		2
▽	ØUT	▽		✓	3
▽	ØUT	▽	+5		4

▽	ØUT	▽	

is the only one that is right, so it gets the tick. All the others are wrong.

Now turn over and do 6 more over the page.

PROGRAM SYNTAX

COLUMN 1	COLUMN 2	COLUMN 3	
	IN		1
	IN	ABC	2
	IN	5	3
	IN	+5	4
	LØAD		1
	+5	LØAD	2
	LØAD	5	3
	LØAD	+5	4
	ADD		1
	ADD	5	2
	ADD	B4D	3
	ADD	+B4D	4
	HALT		1
	HALT	XYZ	2
	HALT	5	3
	HALT	+5	4
	STØRE		1
	STØRE	5	2
	STØRE	+5	3
	STØRE	A	4
	JUMP		1
	JUMP	5	2
	JUMP	+5	3
	JUMP	B5	4

THIS IS THE END OF THE TEST

CESIL CODING SHEET

4/361(8.70)

(60-65)

form B

MASSEY UNIVERSITY

Project PEACE

Programming Skills Test

CODING KNOWLEDGE

Mark 2c

Write your name here: _____

Group Number: _____

Personnel Number: _____

This is a test to see how much you know about the computer language CESIL.

First of all there are 14 programs which have been written out for you, like this:

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C		ØUT PUT
	IN							
	ØUT							
	ØUT							
	HALT							

Data	+5	+7				

You have to fill in the EFFECT side, like this:

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C		ØUT PUT
	IN		5					
	ØUT		5					5
	ØUT		5					5
	HALT							

Data	+5	+7				

Did you see where the output goes?

TURN OVER

Now here is one for you to try:

					EFFECT					
COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C		ØUT PUT
	▽	IN	▽							
	▽	IN	▽							
	▽	IN	▽							
	▽	ØUT	▽							
	▽	HALT	▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							
	▽		▽							
Data		+5		+9	+12		+7			

ASK YOUR TEACHER NOW if there is anything you do not understand. You cannot ask once the test has started.

When you are told, turn this page and start the test.

DO NOT TURN OVER UNTIL YOU ARE TOLD TO START

PROGRAM STØRE

EFFECT

(1) (2) (3)

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUT PUT
	IN								
	STØRE	A							
	IN								
	STØRE	B							
	STØRE	C							
	STØRE	A							
	HALT								
									(4-13)

Data	+8	+4					

Function of Program

PROGRAM LØAD

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STØRE	A							
	IN								
	ØUT								
	LØAD	A							
	ØUT								
	HALT								
									(14-17)

Data	+6	+7					

Function of Program

4.

4/361 (8.70) CESIL CODING SHEET

PROGRAM ADD

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STØRE	A							
	IN								
	ADD	A							
	ØUT								
	HALT								
									(18-19)

Data	+5	+2					

THERE ARE TWO QUESTIONS ON THIS PAGE !!!

PROGRAM MULTIPLY

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STØRE	A							
	IN								
	MULTIPLY	A							
	ØUT								
	HALT								
									(20-22)

Data	+5	+6					

PROGRAM SUBTRACT

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STORE	A							
	IN								
	SUBTRACT	A							
	ØUT								
	HALT								
									(23-25)

Data	+2	+10					

Function of Program

PROGRAM D I V I D E

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A					ØUT PUT
	IN								
	STØRE	A							
	IN								
	DIVIDE	A							
	ØUT								
	HALT								
									(26-28)

Data	+15	+4					

Function of Program

PROGRAM LØADCØNST EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUT PUT
	IN								
	LØAD	+2							
	ØUT								
	HALT								
									(29-31)

Data	+8						

Function of Program

CESIL CODING SHEET

PROGRAM CØNSTS

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUT PUT
	IN								
	STØRE	B							
	ADD	+12							
	ØUT								
	IN								
	SUBTRACT	+2							
	ØUT								
	IN								
	MULTIPLY	+4							
	ØUT								
	HALT								
									(32-42)

Data	+6	+4	+5	+12			

Function of Program

PROGRAM J U M P

EFFECT

COLUMN 1	COLUMN 2	COLUMN 3	ACC.	A	B	C			ØUTPUT
	IN								
	JUMP	B3							
	IN								
B3	ØUT								
	HALT								
	•								
									(43-45)

Data	+8	+2					

Function of Program

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C			OUTPUT
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
	▽	IN	▽								
	▽	ØUT	▽								
	▽	JIZERØ	▽	B4							
B4	▽	HALT	▽								
	▽		▽								(46-56)

Data							
	+4	+7	−2	+0	+5	+8	

ODING SHEET

PROGRAM | J | I | N | E | G | | EFFECT

COLUMN 1		COLUMN 2		COLUMN 3	ACC.	A	B	C			OUTPUT
	▽	IN	▽								
	▽	JINEG	▽	B6							
	▽	IN	▽								
	▽	JINEG	▽	B6							
	▽	IN	▽								
	▽	JINEG	▽	B6							
B6	▽	ØUT	▽								
	▽	HALT	▽								
	▽		▽								(57-59)

Data							
	+8	−3	+7				

On the next page you will find six groups of four CESIL instructions, like this:

	COLUMN 2		COLUMN 3	
▽	ØUT	▽	PQ	
▽	PQ	▽	ØUT	
▽	ØUT	▽		
▽	ØUT	▽	+5	

In each group, only <u>one</u> instruction is written correctly. Put a tick beside the one that is <u>correct</u>, like this:

▽	ØUT	▽	PQ	
▽	PQ	▽	ØUT	
▽	ØUT	▽		✓
▽	ØUT	▽	+5	

▽	ØUT	▽	

is the only one that is right, so it gets the tick. All the others are wrong.

Now turn over and do 6 more over the page.

PROGRAM SYNTAX

COLUMN 1	COLUMN 2	COLUMN 3
	IN	JIM
	IN	8
	IN	+8
	IN	
	HALT	PETE
	HALT	
	HALT	B8
	HALT	+8
	STØRE	PETE
	STØRE	+8
	STØRE	8
	STØRE	
	JUMP	8
	JUMP	+8
	JUMP	B8
	JUMP	+B8
	LØAD	+JIM
	JIM	LØAD
	LØAD	JIM
	LØAD	
	ADD	PETE
	ADD	+PETE
	ADD	8
	ADD	

(60-65)

Standardisation of the test

ALGORITHMIC SKILL Mark 1

ADMINISTRATION

Each child needs a pencil and an eraser. Pupils must be so placed as to make cooperative work impossible. The whole testing procedure takes half an hour, so ensure that you are free of interruptions for this period of time. It is absolutely essential that the children are not disturbed during the test itself.

Test booklets are distributed. The teacher says:

Keep your booklet closed. Write your name on the front. Also write your group number and personnel number. When everyone has done that, we'll open the booklets.

This done, the teacher says:

Now open your booklet and look at page 2. You will see a flowchart there, and on the opposite page there is a grid of lines. The flowchart tells you what you have to do on the grid. In this one, it goes like this.

The teacher works the problem on the blackboard, taking care to emphasize what happens to the number "N" in the process. This should produce the same result as is printed in page 3. The teacher then says:

Each time you do one of these questions, you get a pattern on the right-hand page. Now turn over and you try the next one.

When the children have done so, the teacher says:

Did you turn over to the next page as the flowchart told you to? You can see there the kind of pattern you should have got that time. Did you understand the bit about the squares and circles?

The teacher gives such assistance as is required to ensure that all children understand how to do problem 00. Pages 6 and 7 are then discussed in detail to ensure that they are understood. The teacher says:

Don't turn over yet. Is there anything you're not clear about? (deals with queries). In a minute, when I tell you to start, you will have twelve minutes to do as many questions as you can. Don't leave any questions out - do every one you come to. Ready? Go.

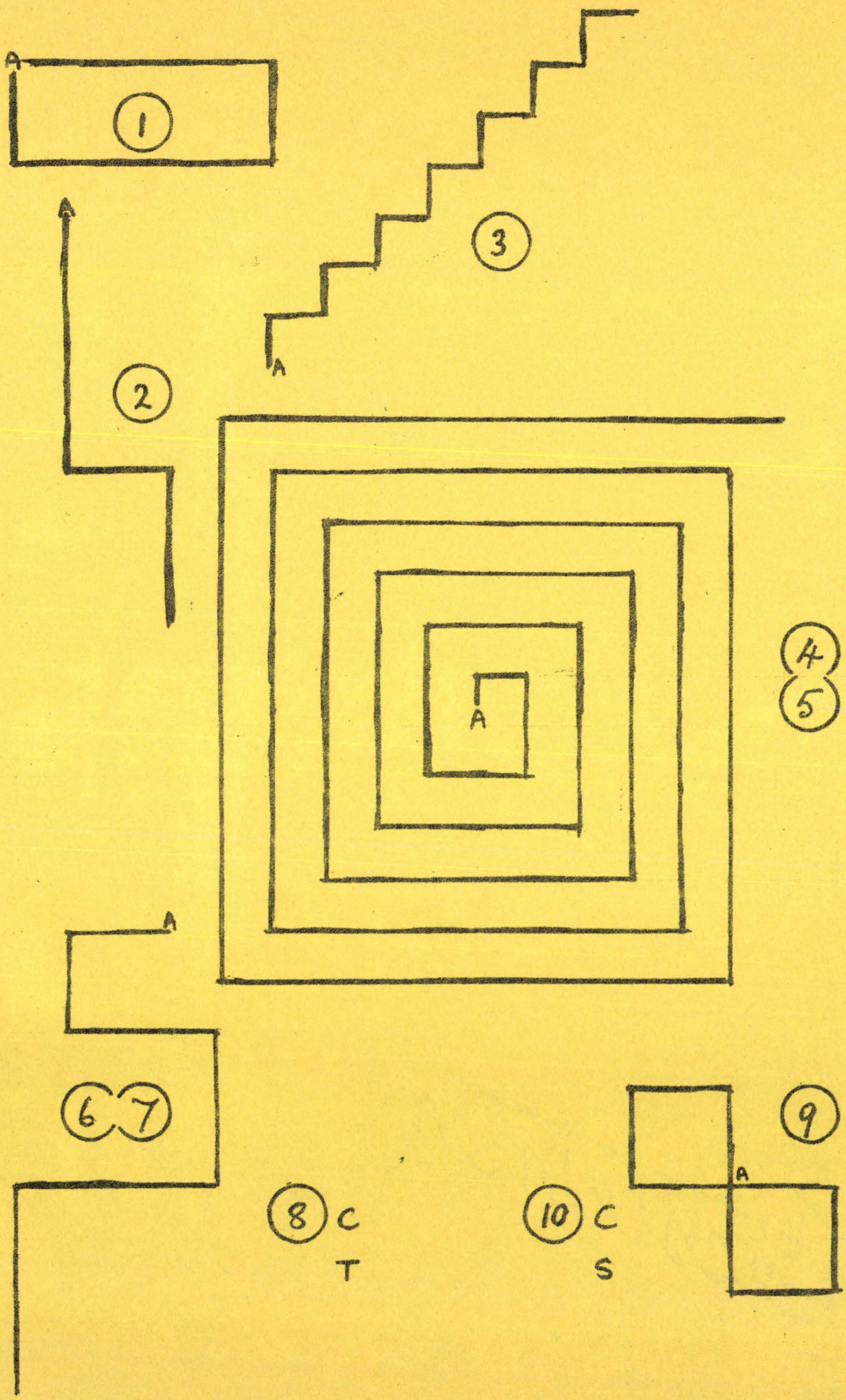
The test proceeds for exactly 12 minutes. During this time the only help the teacher can give pupils is in matters relating to possible misprints in the test booklet, etc. When 12 minutes have elapsed, the teacher says:

Stop. Pencils down. Close your booklets.

NOTES ON SCORING "ALGORITHMIC SKILL MK 1"

	For 2 points	For 1 point
1.	Exactly as per model. Rectangle must be correct size, shape and position.	Not possible to obtain 1 point on this item
2.	Exactly as per model	Not possible to obtain 1 point on this item
3.	As per model. Last stroke must be horizontal, finish at top left-hand corner.	Correct staircase formed but not finished correctly.
4.	As per model. Last stroke must be horizontal, finish at top left-hand corner.	(i) Correct Spiral formed but not finished correctly. Alternatively, (ii) some spiral similar to the correct one, but incorrect as the result of a single mistake; must be correctly finished to score.
5.	(i) Answer identical to whatever was offered for 4. The point he e is that the respondent perceives that this algorithm is equivalent to the previous one. Alternatively (ii) as per model.	- As for 4 -
6.	As per model.	(i) Final stroke wrong or missing but otherwise correct. Alternatively (ii) any one stroke of incorrect length but correct direction.
7.	(i) Answer identical to that offered for 6. Alternatively (ii) as per model.	- As for 6 -
9.	As per model	One square completed correctly but remainder either wrong or missing
8 & 10.	Both items answered correctly.	One item answered correctly.

(N.B.) Any method of indicating an answer is accepted if it is u ambiguous)


A
1
3
A
2
A
4
5
6
7
8
C
T
10
C
S
9
A

MASSEY UNIVERSITY

Project PEACE

ALGORITHMIC SKILL

Geometric Patterns

Mark 1c

Write your name here _____

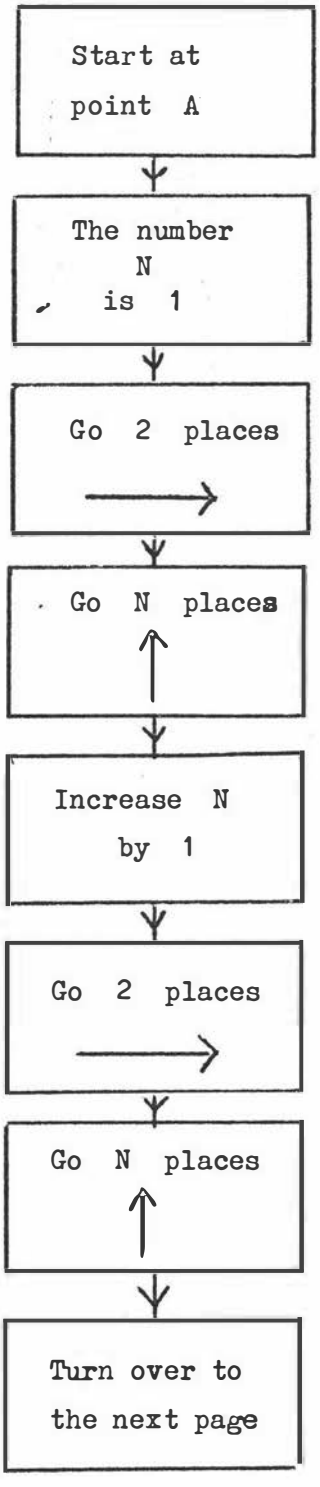
Group number _____

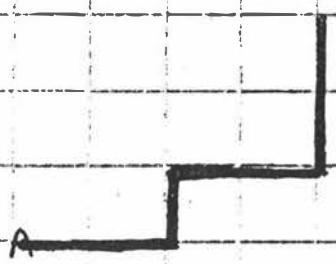
Personnel number _____

HOW TO DO THIS TEST

In this test you will find a lot of flowcharts. Most times, the flowchart is on the left hand page (like this one), and on the right hand page there is a grid. You follow the flowchart, marking out a pattern on the grid with your pencil. This one has been done already : follow it yourself.

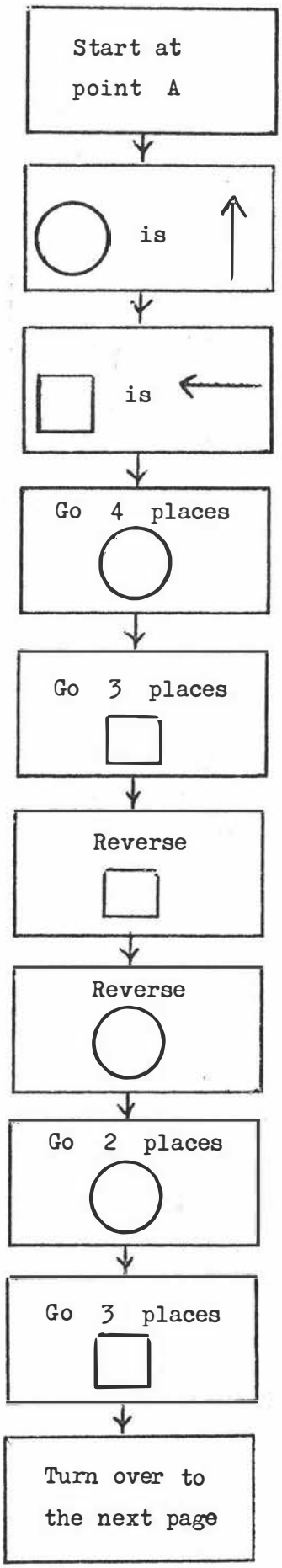
(0)




A

Now here is one for you to do yourself. It is a little bit different, but see if you can do it.

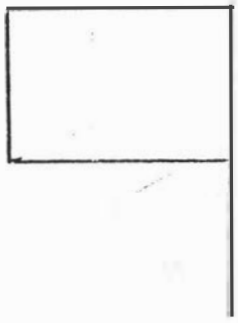
(00)



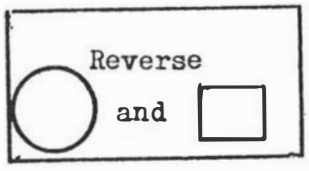
00

A

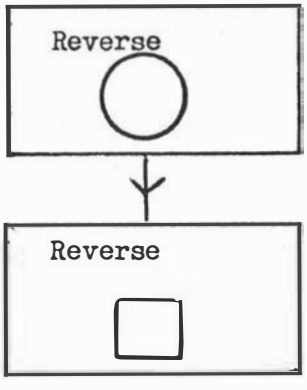
If you did that right, you should have made a shape like this :



Sometimes you will meet



which is just a neater way of showing

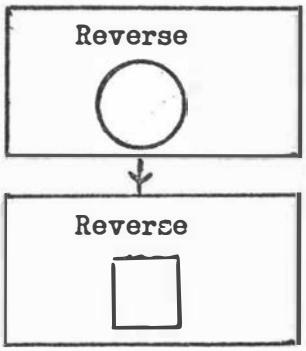


DO YOU UNDERSTAND ALL THAT ?

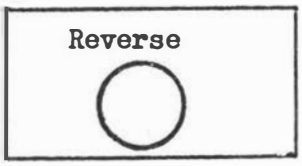
IF YOU DON'T, <u>ASK YOUR TEACHER NOW</u>.

There are a few questions where you have a choice of answers, like this:

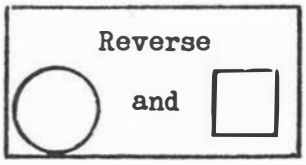
Which of these means exactly the same as



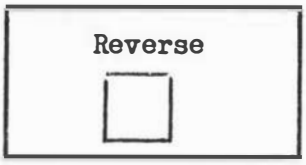
A



B



C



Your choice: ABC

Well, the answer is B, so cross out the A and the C so the answer looks like this:

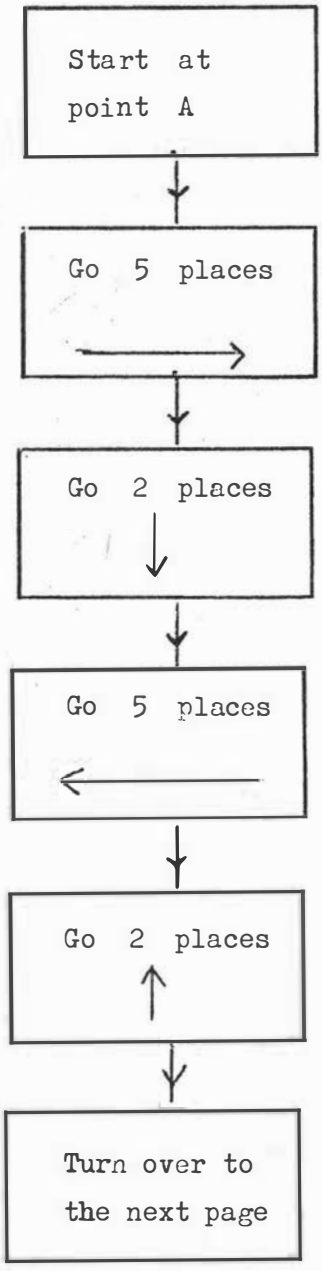
Your choice: ~~A~~B~~C~~

DO YOU UNDERSTAND IT ALL NOW? IF YOU DON'T, ASK YOUR TEACHER NOW

You can't ask questions once the test has started.

DON'T TURN THIS PAGE UNTIL YOU ARE TOLD TO

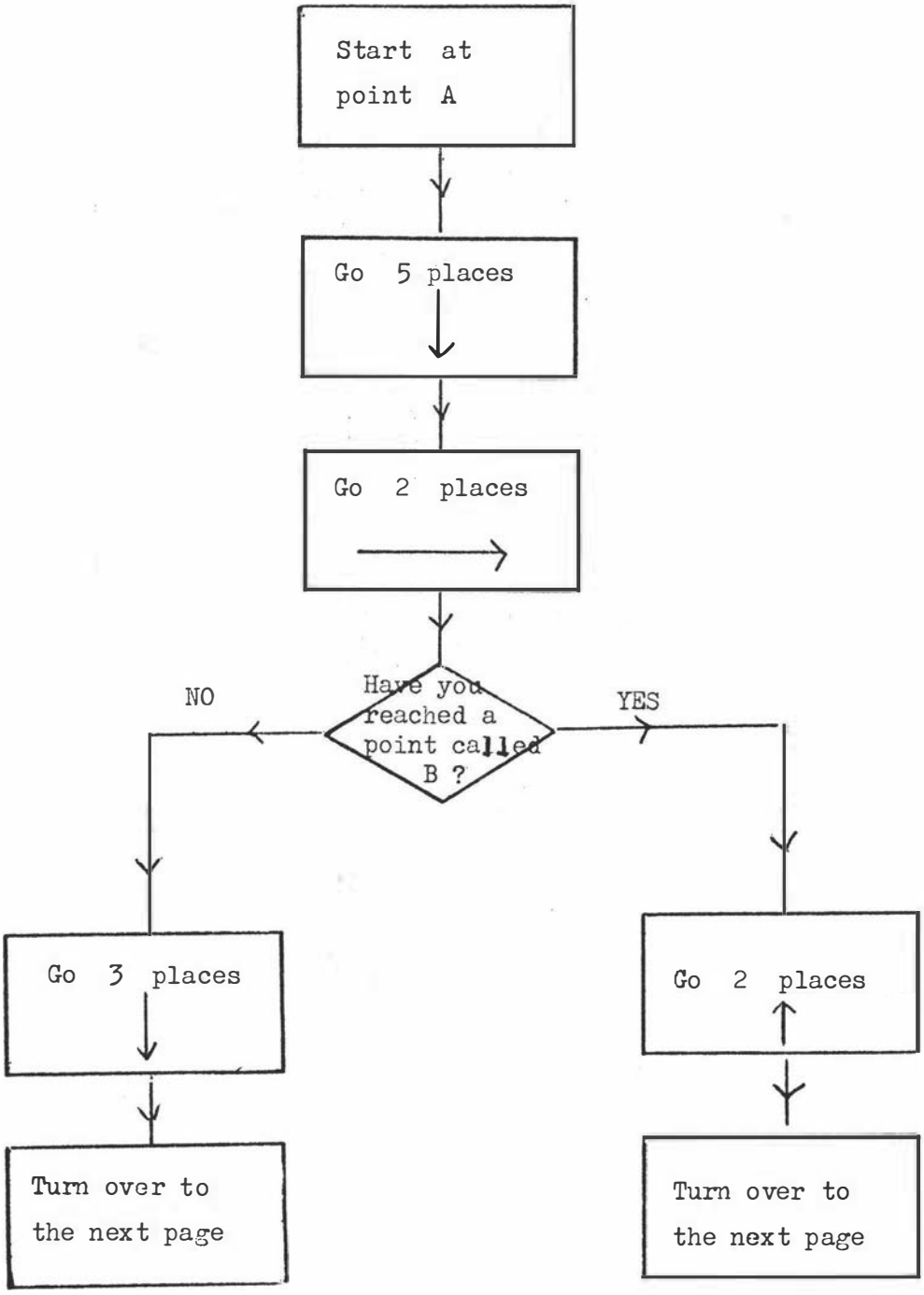
(1.)

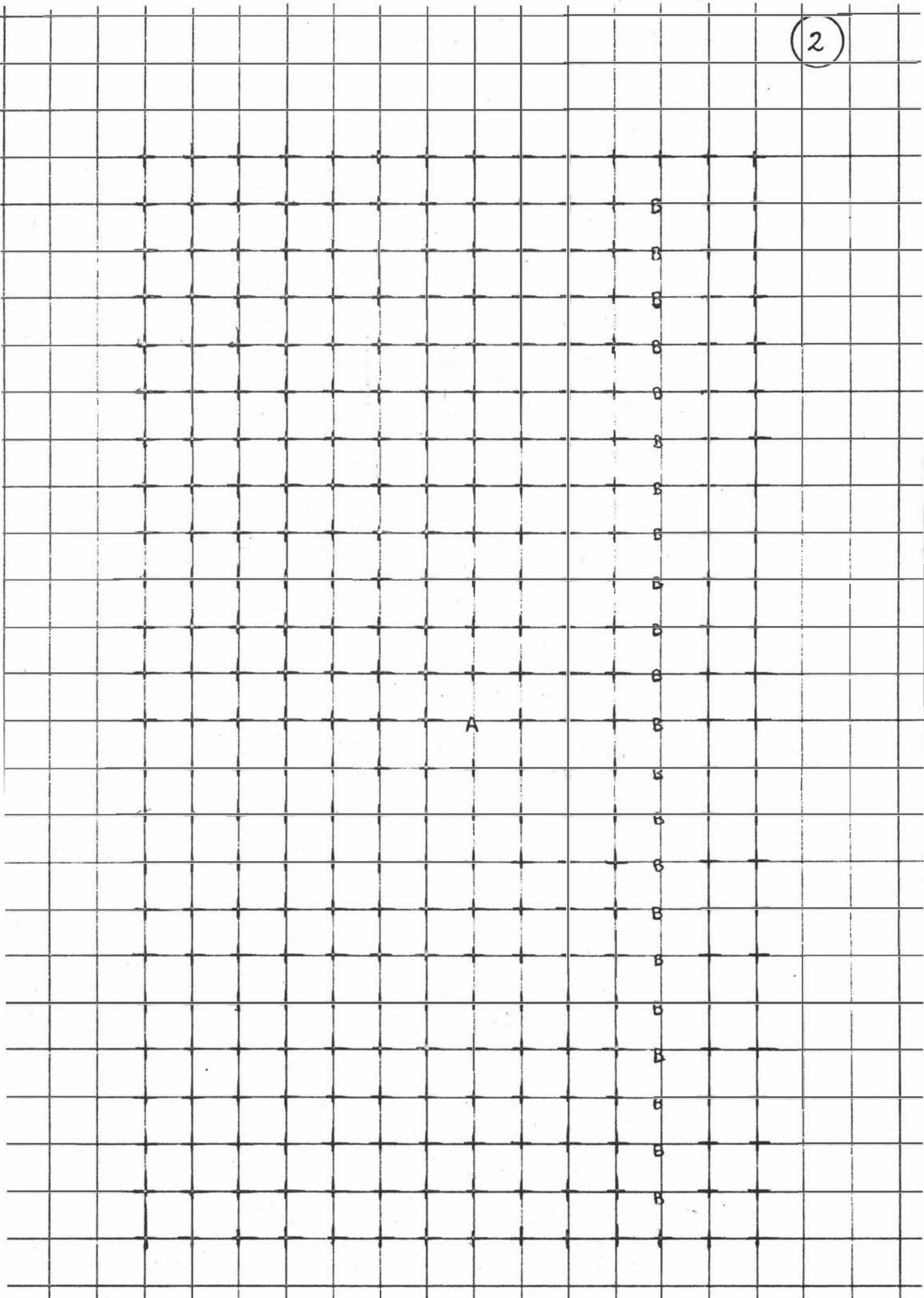

Start at point A
Go 5 places
Go 2 places
Go 5 places
Go 2 places
Turn over to the next page

(1)

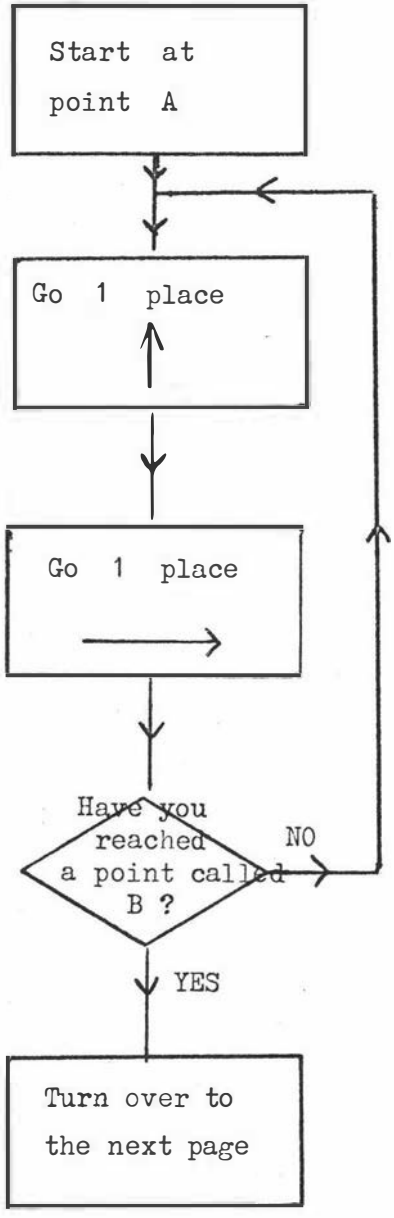
A

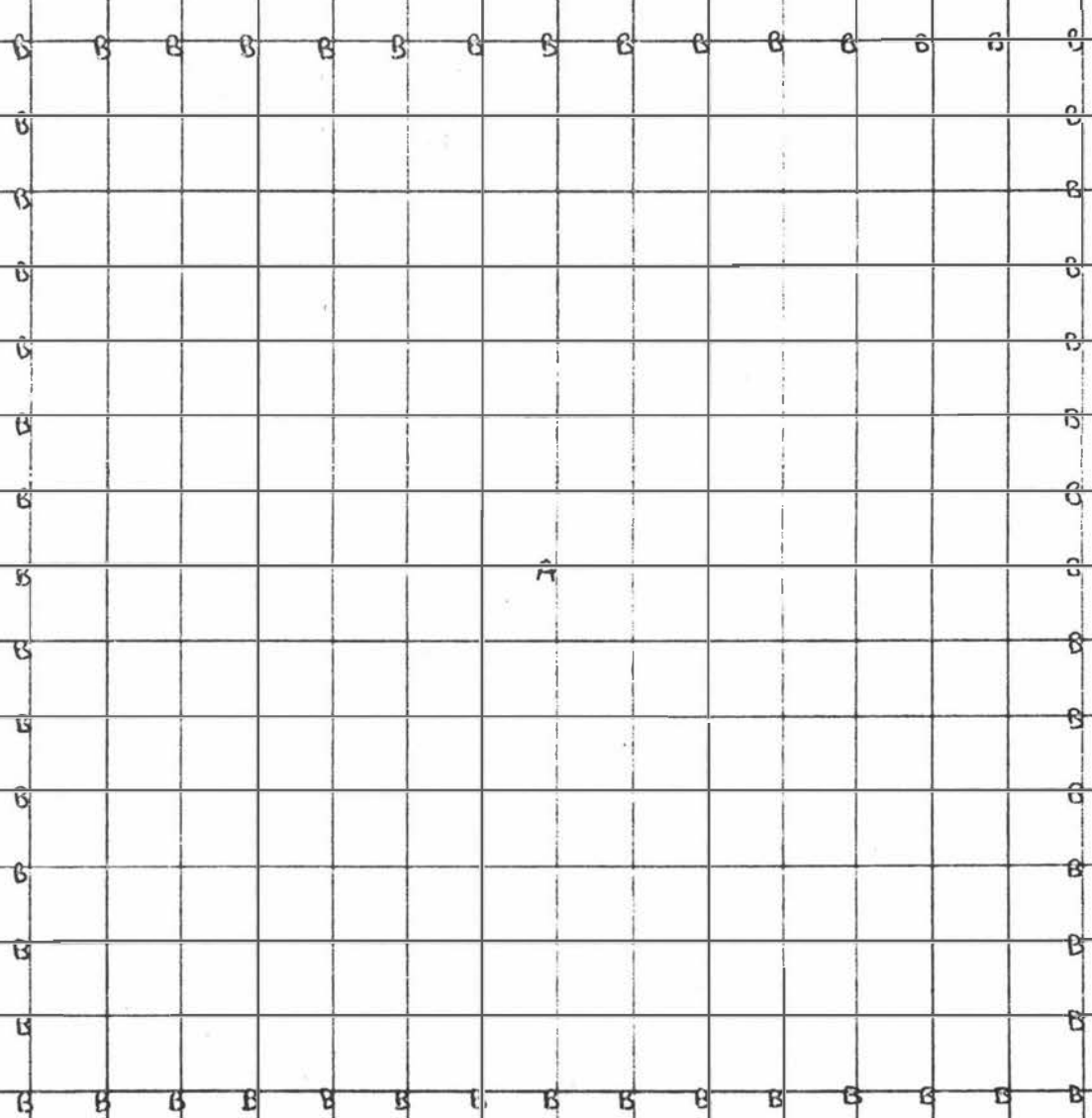
(2.)


Start at point A
Go 5 places
Go 2 places
Have you reached a point called B ?
NO
YES
Go 3 places
Turn over to the next page
Go 2 places
Turn over to the next page

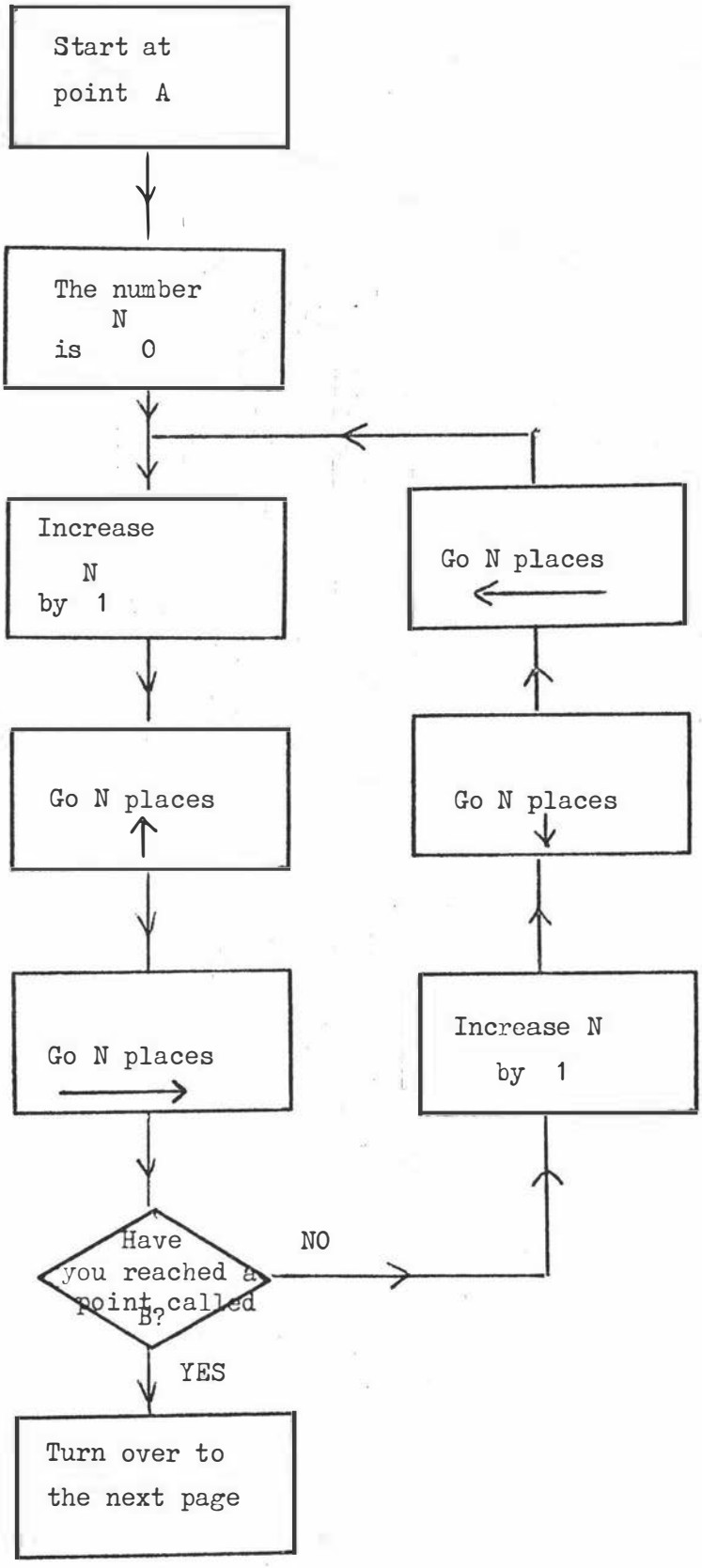

A
B

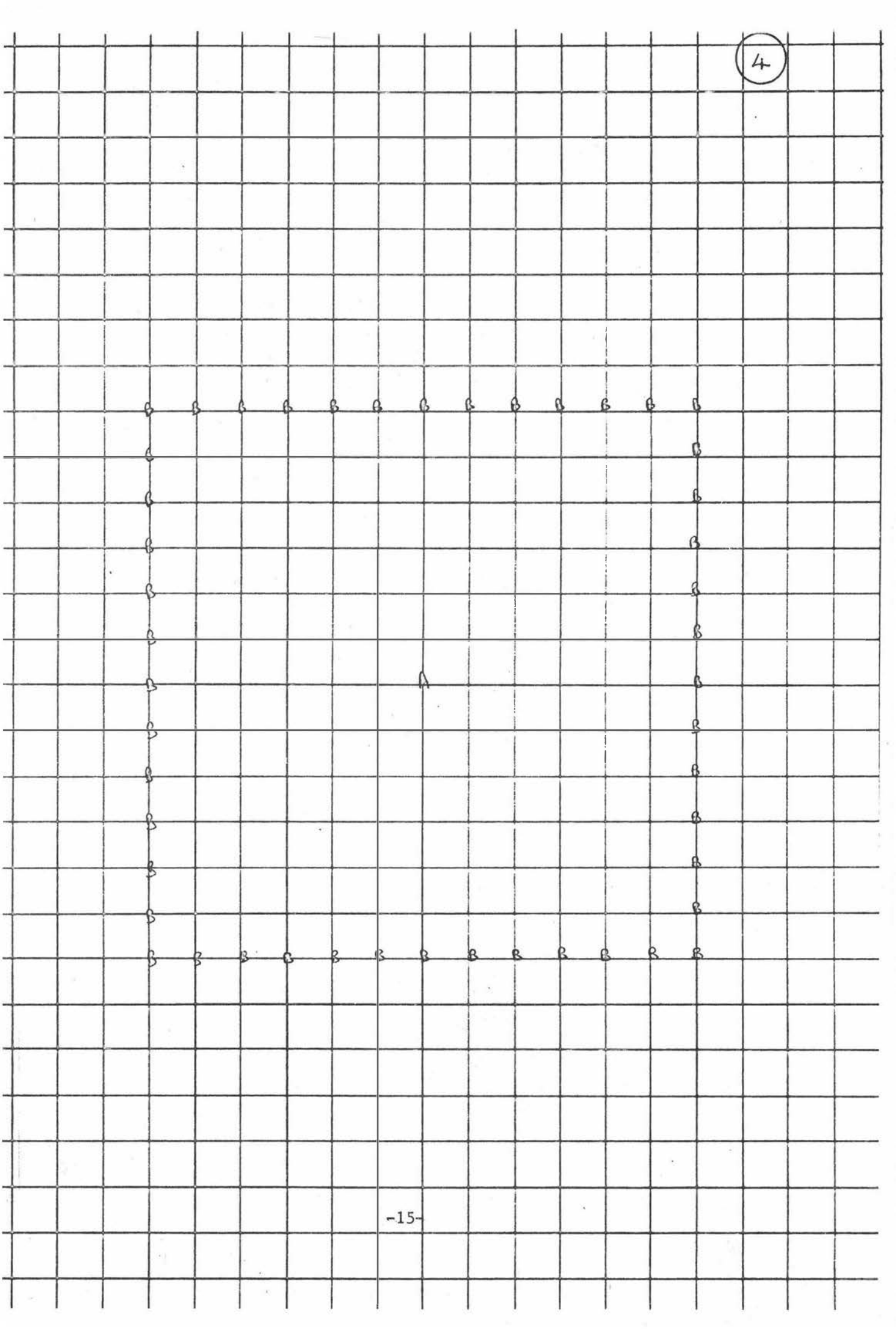
3.


Start at point A
Go 1 place
Go 1 place
Have you reached a point called B ?
NO
YES
Turn over to the next page

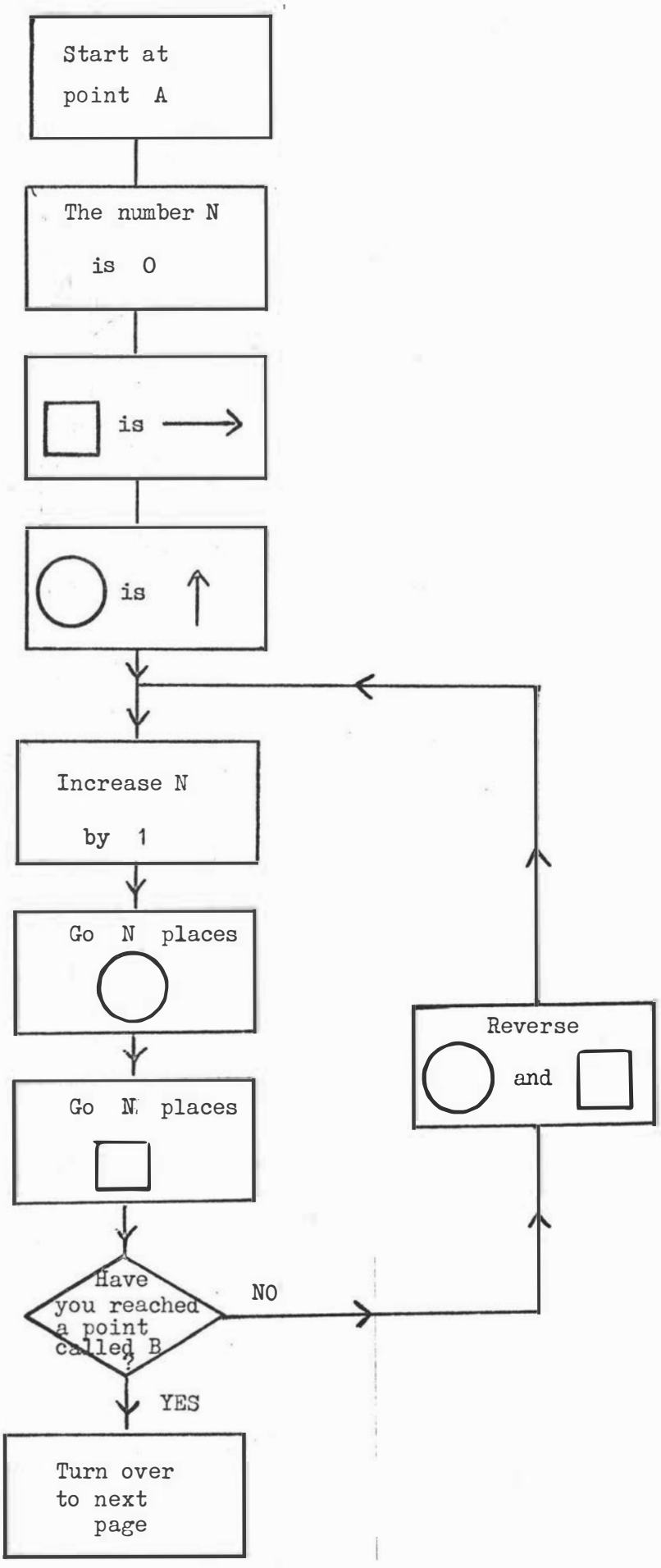


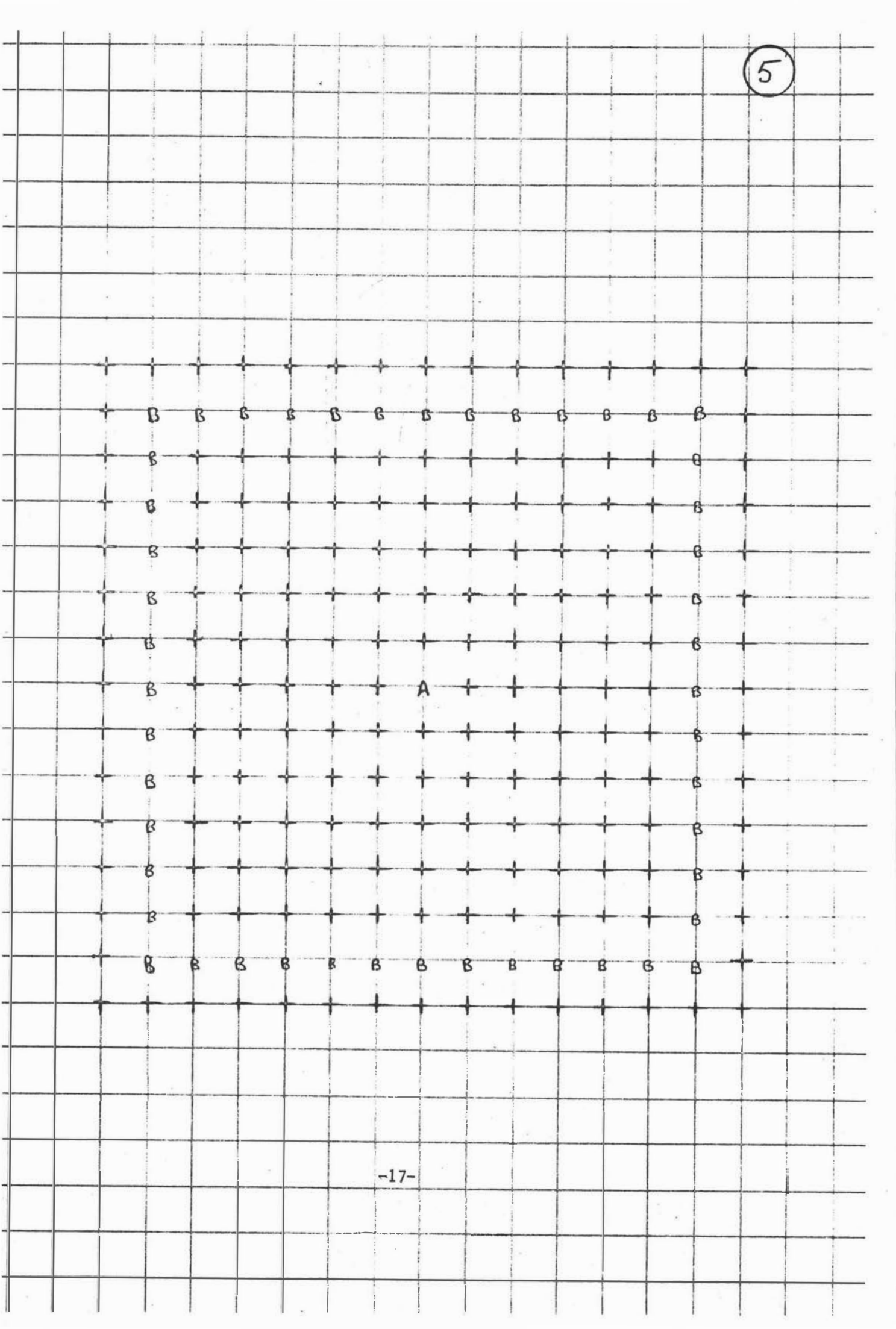
(4.)


Start at point A
The number N is 0
Increase N by 1
Go N places
Go N places
Go N places
Go N places
Increase N by 1
Have you reached a point called B?
NO
YES
Turn over to the next page

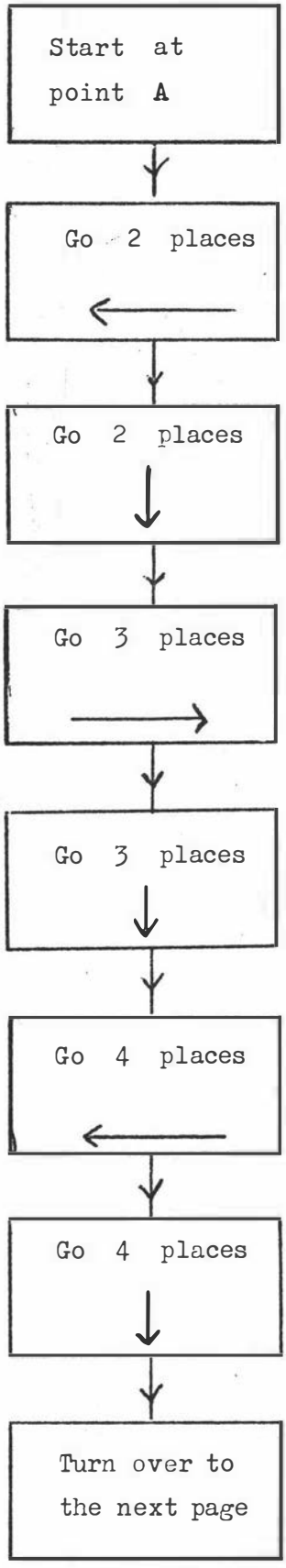

4

(5.)

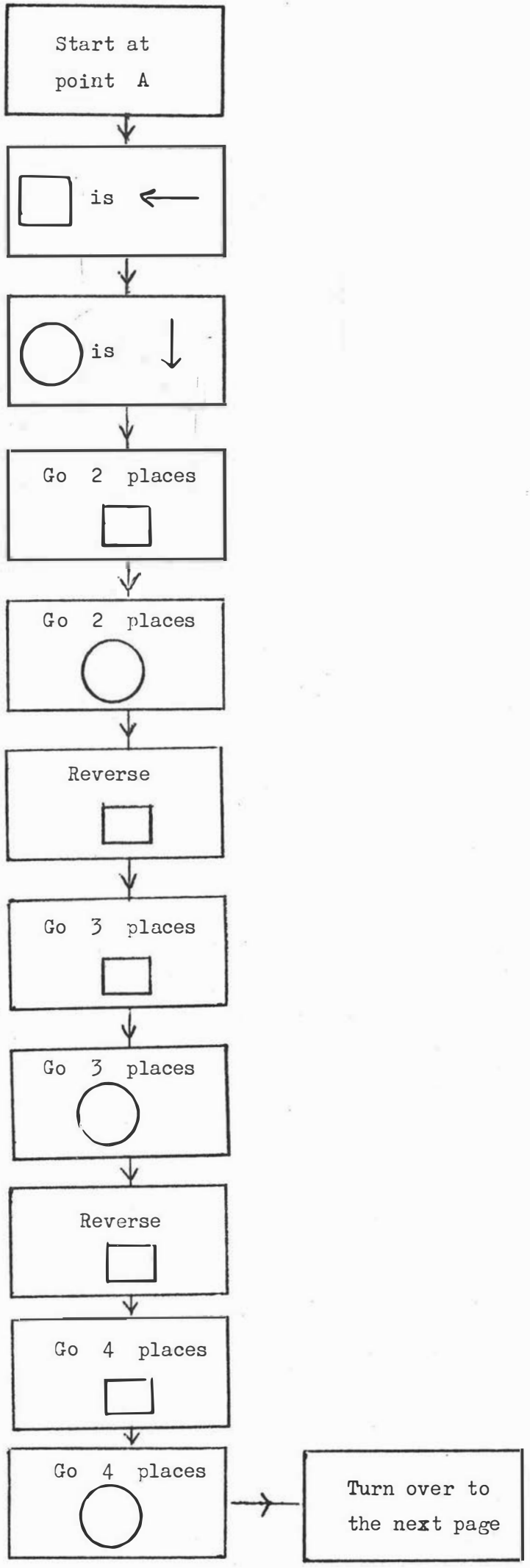

Start at point A
The number N is 0
is
is
Increase N by 1
Go N places
Go N places
Have you reached a point called B ?
NO
YES
Reverse and
Turn over to next page


5
A

(b.)


Start at point A
Go 2 places
Go 2 places
Go 3 places
Go 3 places
Go 4 places
Go 4 places
Turn over to the next page

7.


Start at point A
is
is
Go 2 places
Go 2 places
Reverse
Go 3 places
Go 3 places
Reverse
Go 4 places
Go 4 places
Turn over to the next page

7

A

(8.)

The flowchart on the opposite page is supposed to give the same result as number (6.) But there are two empty boxes which need to be filled to finish the flowchart off.

Your job is to decide which of the boxes below should go in the places x and y so that this flowchart does do the same job as number (6).

The box  should be

A

The number N is zero

B

The number N is 1

C

The number N is 2

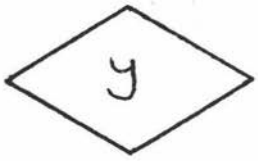
The number N is 4

D

The number N is 5

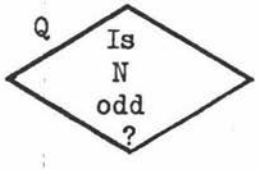
E

Your Choice: ABCDE

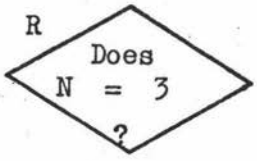
The diamond  should be

P

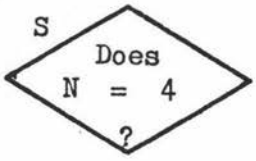
Q



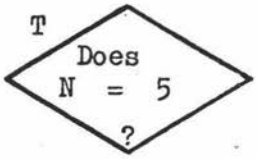
R



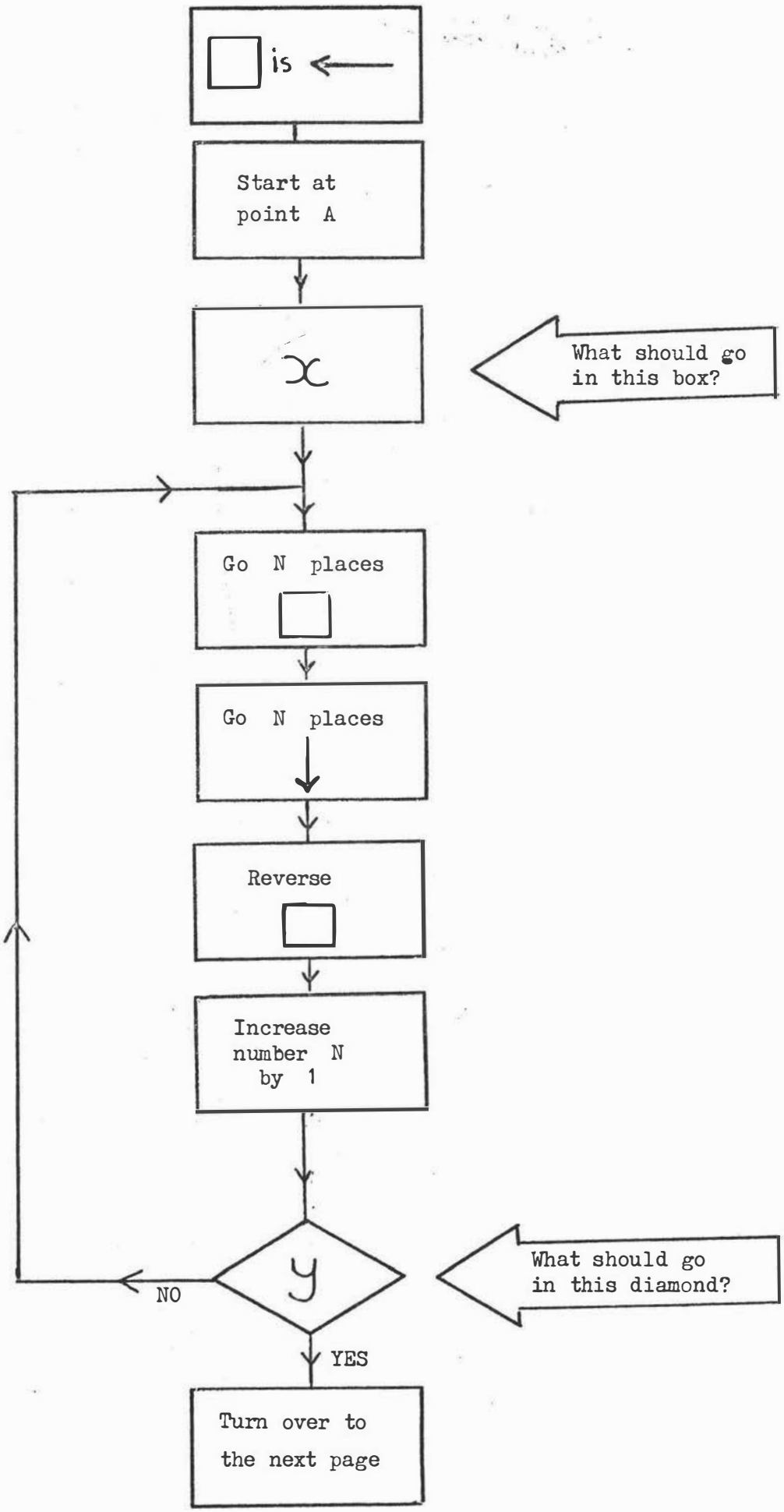
S



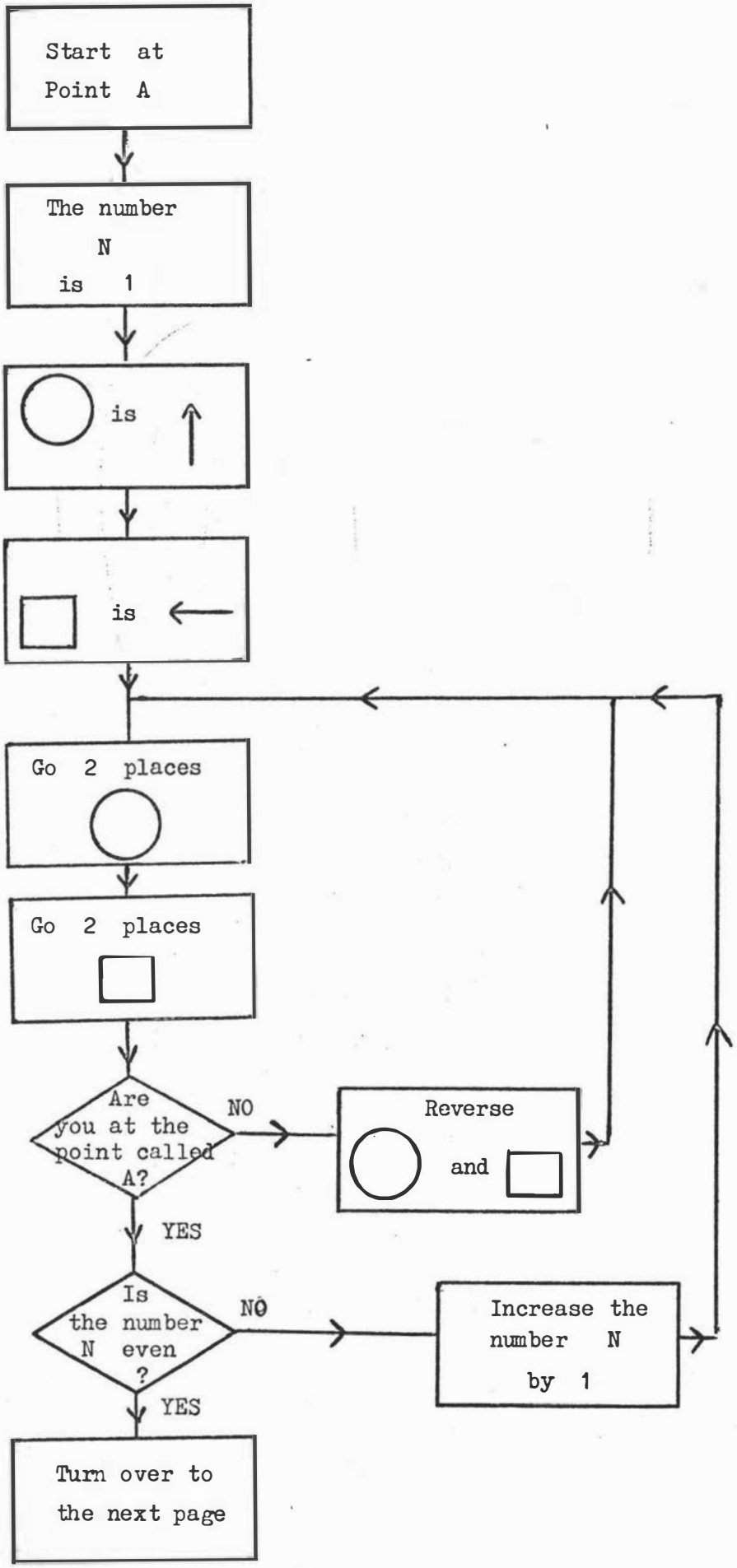
T



Your Choice: PQRST

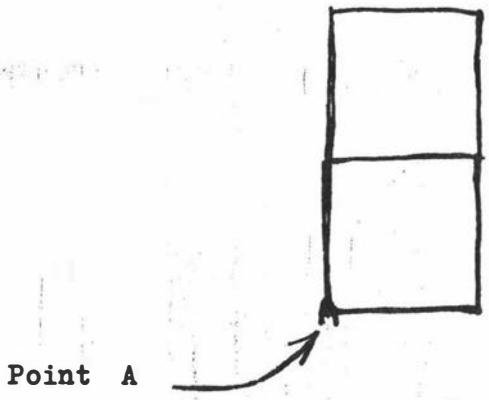

□ is ←
Start at point A
x
What should go in this box?
Go N places
□
Go N places
↓
Reverse
□
Increase number N by 1
y
What should go in this diamond?
NO
YES
Turn over to the next page

(9.)


Start at Point A
The number N is 1
is
is
Go 2 places
Go 2 places
Are you at the point called A?
NO
Reverse
and
YES
Is the number N even ?
NO
Increase the number N by 1
YES
Turn over to the next page

A

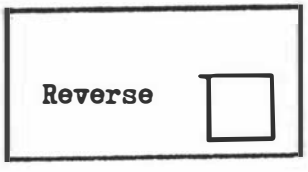
The flowchart on the opposite page is supposed to give a result like this:



Your job is to decide which of the boxes below should go in the places S and t so that this flowchart does give this pattern.

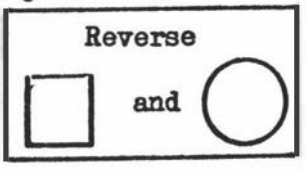
The box  should be

A



B

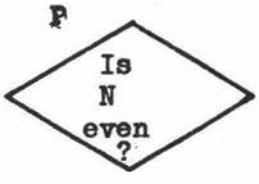
C



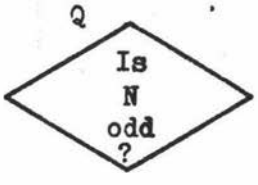
Your Choice: ABC

The diamond  should be

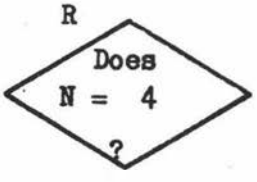
P



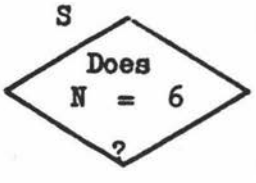
Q



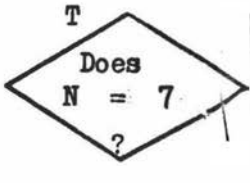
R



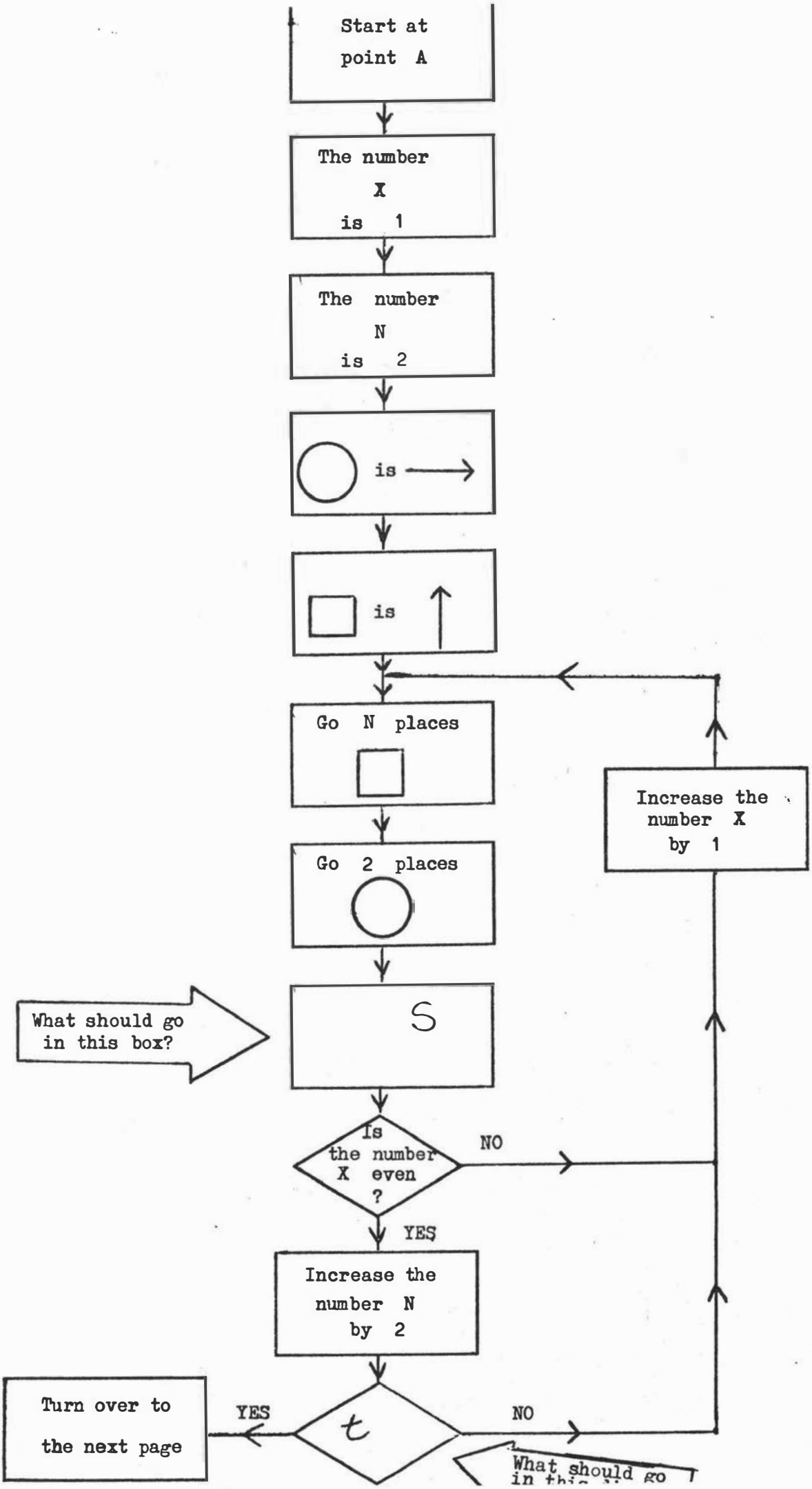
S



T



Your Choice: PQRST


Start at point A
The number X is 1
The number N is 2
is
is
Go N places
Go 2 places
Increase the number X by 1
What should go in this box?
S
Is the number X even ?
NO
YES
Increase the number N by 2
Turn over to the next page
YES
t
NO
What should go

THIS IS THE END OF THE TEST

BIBLIOGRAPHY

Works Cited

Alspeugh, Carol A. "Identification of some Components of Computer Programming Aptitude." Journal for Research in Mathematics Education, 3, (March 1972), 89 - 98.

Alspaugh, John W. "The Relationship of Grade Placement to Programming Aptitude and FORTRAN Programming Achievement." Journal for Research in Mathematics Education, 2 (Jan 1971), 44 - 49.

Australian Council for Educational Research, ACER Arithmetic Tests, undated.

Bauer, R.; Mehrens, W.A.; and Vinsonhaler, J.F. "Predicting Performance in a Computer Programming Course." Educational and Psychological Measurement, 28 (w.1968), 1159 - 1164.

Bitter, G.G. "Effect of Computer Applications on Achievement in a College Introductory Calculus Course." Unpublished doctoral dissertation, University of Denver, 1970. (Quoted in Hatfield and Kieren (1972))

Cattell, R.B., and Cattell, D.L. Jr - Sr High School Personality Questionnaire. Champaign, Ill. :Institute for Personality and Ability Testing, 1968.

_____. Manual for the Jr - Sr High School Personality Questionnaire. Champaign, Ill.: Institute for Personality and Ability Testing, 1968.

Clutterham, D.R. A Method for Evaluating Student Progress in Undergraduate Computer Science by Use of Automated Problem Sets. Final Report. Washington: Office of Education, 1970.

Consultative Committee on the Curriculum. Computers and the Schools (Bellis Report). Edinburgh: Scottish Education Department, undated.

Cox, B.G. "On the Teaching of Programming at Universities." Proc. Third National Conference of the New Zealand Computer Society. Palmerston North: N.Z.C.S., 1972; 87 - 88.

Darby, C.A.; Korotkin, A.L.; and Komashko, T. Survey of Computing in Secondary Schools. Washington: American Institutes for Research, 1970.

Ebel, R. Essentials of Educational Measurement. Inglewood Cliffs, N.J.: Prentice-Hall, 1972, 419 - 420.

Elley, W.B., and Reid, N.A. Progressive Achievement Tests: Reading Comprehension and Reading Vocabulary. Wellington: N.Z.C.E.R, 1969.

Fielder, A.L. "A Comparison of Achievement Resulting from Learning Mathematical Concepts by Computer Programming versus Class Assignment Approach." Unpublished doctoral dissertation, University of Oklahoma, 1969. (Quoted in Hatfield and Kieren (1972).)

Fieldhouse, A.E. Manual of Directions for the ACER Arithmetic Tests. Wellington: N.Z.C.E.R, 1956.

Hatfield, L.L. Computer Mediated Instruction in Mathematics - Grade 7. Preliminary Report. Minneapolis: University of Minnesota, 1966.

_____, and Kieren, T.E. "Computer Assisted Problem Solving in School Mathematics." Unpublished research report, University of Minnesota, 1971.

_____. "Computer-Assisted Problem Solving in School Mathematics." Journal for Research in Mathematics Education, 3 (March 1972), 99 - 112

Haven, R.N. "For Project LOCAL (Laboratory Program for Computer-Assisted Learning). "Unpublished report, Westwood, Massachusetts Public Schools, 1968. (Quoted in Hatfield and Kieren. (1972).)

Holoien, M.O. "Calculus and Computing: a Comparative Study of the Effectiveness of Computer Programming as an Aid in Learning Selected Concepts in First-Year Calculus." Unpublished doctoral dissertation, University of Minnesota, 1970 (Quoted in Hatfield and Kieren (1970)

Howell, M. A.; Vincent, J.W.; and Gay, R.A. "Testing Aptitude for Computer Programming." Psychological Reports, 20 (June 1967) 1251 - 1256.

International Business Machines Corporation. Aptitude Test for Programmer Personnel. Chicago: IBM, 1964.

_____. Manual for Administering and Scoring the Aptitude Test for Programmer Personnel. White Plains: IBM, 1964

ICL/CES (International Computers Limited/Computer Education in Schools). Computer Studies 1. London: ICL, 1971.

_____. "Computer Studies - What Value to the Pupil?" Unpublished report on teacher attitudes, London, 1972.

Johnson, D.C., et al. Computer Assisted Mathematics Program. Glenview, Ill.: Scott, Foresman , 1968 - .

Katz, A. "Prediction of Success in Automatic Data Processing Course". U.S. Army Personnel Research Office Technical Research Note, 126 (1962). (Quoted in Bauer, et al. (1968).)

Knuth, D.E. The Art of Computer Programming, Reading, Mass: Addison Wesley, 1968, vol.1, 4 - 6

Martin, R.C.L. "Programming as a Communication Activity." Proc. Third National Conference of the New Zealand Computer Society. Palmerston North: NZCS, 1972; 9 - 28.

Mehrens, W.A. and Lehmann, I.J. Measurement and Evaluation in Education and Psychology. New York: Holt, Rinehart, Winston, 1973, 113.

Morris, J.L., and Parkinson, M. "Vocational Interests of Data Processing Personnel". Australian Psychologist, 6 (1971), no. 6, 19 - 25.

_____, and Wise, J.J. "Personality Characteristics of Data Processing Personnel". Australian Psychologist, 7 (1972), no. 3, 173 - 179.

Nie, Bent and Hull. Statistical Package for the Social Sciences. New York: McGraw-Hill, 1970

NZCER. Manual for the Otis Intermediate Test Wellington: NZCER, 1969.

OECD Centre for Educational Research and Innovation. Report. Paris: OECD, 1971.

Otis, A.S. Otis Self-Administering Test. Intermediate Examination. Wellington: NZCER, undated.

Palormo, J.M. Computer Programmer Aptitude Battery. Chicago: Science Research Associates, 1964.

_____. Computer Programmer Aptitude Battery: Manual. Chicago: Science Research Associates, 1967.

_____. "The Computer Programmer Aptitude Battery - a Description and discussion." Proc. Fifth Annual Computer Personnel Research Conference. New York: Assn. for Computing Machinery, 1967.

Papert, S. "Teaching Children Thinking." Unpublished report, M.I.T. Artificial Intelligence Laboratory (Memo. no. 247), 1971.

Reed Mathematics Group. The Shape of Mathematics - 2A. Wellington: Reed, 1970.

_____. Units in Mathematics. T7: Formulae. Wellington: Reed, 1973.

Reid, N.A., and Hughes, D.C. Progressive Achievement Tests: Mathematics. Wellington: NZCER, 1974.

_____. Progressive Achievement Tests: Mathematics. Test Manual. Wellington: NZCER, 1974.

Rowan, T.C. "Psychological Tests and Selection of Computer Prog rammers." Journal of the Assn, for Computing Machinery; 4 (1957), 348 - 353.

Seiler, J. "Abilities for ADP Occupations." Proc. Third Annual Computer Personnel Research Conference. New York: Assn for Computing Machinery, 1965.

Smith, L., and Smith, C. Computer Age Mathematics: Form 1. Sydney: Jacaranda, 1971; preface.

Software Writing Group, Programming Dept., Digital Equipment Corp. Introduction to Programming. Maynard, Mass.: DEC, 1970, 3-2,3-3.

Spence, R.J. "Delimiting Factors in Computer Education at the Secondary School Level." Proc. Second National Conference of the New Zealand Computer Society. Dunedin: NZCS, 1970, 73 - 90.

_____. Multiple Choice Tests in Mathematics for Junior Forms. Wellington: Reed, 1973.

_____. "CBS Among the Kiwis." Computer Education in Schools Newsletter, 6 (December 1974); London: ICL/CBS.

_____. "Computer: the Marvellous Toy." Education, 24 (1975) no. 2, 3 - 7.

Stalnaker, A.W. "The Watson-Glaser Critical Thinking Appraisal as a Predictor of Programming Performance." Proc. Third Annual Computer Personnel Research Conference. New York: Assn for Computing Machinery, 1965. (Quoted in Alspaugh, C.A. (1972).)

Wallace, D.C. "The Impact of Computer Mathematics on the Learning of High School Trigonometry and Physics." Unpublished doctoral dissertation, University of Texas, 1968. (Quoted in Hatfield and Kieren (1972).)

Weinberg, G.M. The Psychology of Computer Programming. New York: van Nostrand, Reinhold, 1971.

Wolfe, J.M. "Perspectives on Testing for Programming Aptitude." Proc. 1971 Annual Conference of the Assn for Computing Machinery, Chicago: ACM, 1971, 268 - 277.

_____. Wolfe Programming Aptitude Test (Experimental Form S). New York: Programming Specialists, 1972.

_____. "Women Are More Predictable than Men." Datamation, Feb. 1974, 67 - 70.