IN PROGRAMMING

Technical Report AIP - 135

Quanfeng Wu and John R. Anderson
Department of Psychology

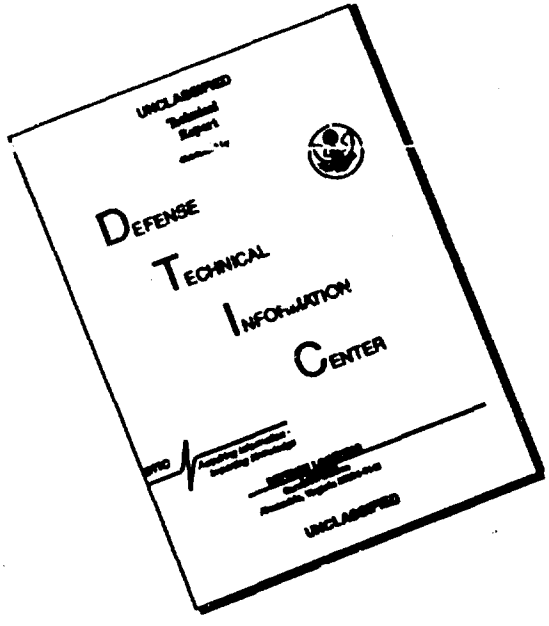
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

(2)

STRATEGY CHOICE AND CHANGE IN PROGRAMMING

Technical Report AIP - 135

Quanfeng Wu and John R. Anderson
Department of Psychology

Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.

May 1, 1990

This research was partially supported by the Computer Science Division, Office of Naval Research, under contract number N00014-86-K-0678. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 135	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research (Code 1133)
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, PA 15213	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p40005ub201/7-4-86 PROGRAM ELEMENT NO: N/A; PROJECT NO: N/A; TASK NO: N/A; WORK UNIT ACCESSION NO: N/A

11. TITLE (Include Security Classification)
Strategy choice and change in programming

12. PERSONAL AUTHOR(S) Quanfeng Wu and John R. Anderson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	PASCAL, programming languages, human-computer interaction

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

SEE REVERSE SIDE

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b. TELEPHONE (Include Area Code) (202) 696-4302 22c. OFFICE SYMBOL N00014

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

This research studied iterative or looping strategy choices and changes, especially between the "**while-do**" and "**repeat-until**" looping constructs in the PASCAL programming language. The empirical results from the first experiment, in which subjects were free to choose between the two looping alternatives, indicated that most of PASCAL programmers are quite sensitive to the nature of the problems being solved and adaptable in choosing appropriate looping strategies. Another two experiments were performed in which subjects were either forced or induced to use one of the two looping strategies. These two experiments indicated that subjects are quite tenacious in using the appropriate strategy and their performance deteriorates when they are forced to use a different strategy. These results are consistent with results of Reder (1987, 1988), Reder and Ritter (1990), and Siegler and Jenkins (1989) on strategy selection in other domains and with other populations

Strategy Choice and Change in Programming

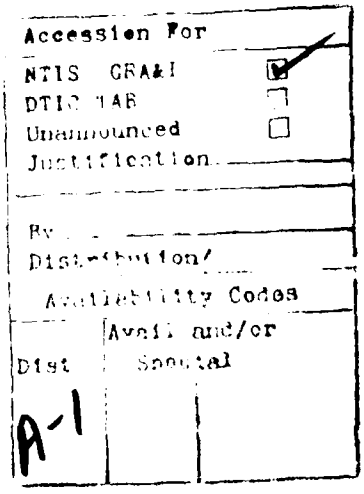
Quanfeng Wu & John R. Anderson

Department of Psychology

Carnegie Mellon University

Pittsburgh, PA 15213

May 1, 1990



Iterative Strategy Choices and Changes in PASCAL Programming

QUANFENG WU AND JOHN R. ANDERSON
Carnegie Mellon University

This research studied iterative or looping strategy choices and changes, especially between the "**while-do**" and "**repeat-until**" looping constructs in the PASCAL programming language. The empirical results from the first experiment, in which subjects were free to choose between the two looping alternatives, indicated that most of PASCAL programmers are quite sensitive to the nature of the problems being solved and adaptable in choosing appropriate looping strategies. Another two experiments were performed in which subjects were either forced or induced to use one of the two looping strategies. These two experiments indicated that subjects are quite tenacious in using the appropriate strategy and their performance deteriorates when they are forced to use a different strategy. These results are consistent with results of Reder (1987, 1988), Reder and Ritter (1990), and Siegler and Jenkins (1989) on strategy selection in other domains and with other populations.

We are very grateful to Drs. Herbert Simon and Kurt VanLehn, and to many graduate students in the Department of Psychology at CMU, for their comments on various portions of this work. Reprint requests should be sent to John R. Anderson, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213.

INTRODUCTION

The issue of cognitive strategy choice is an important research topic in cognitive psychology in that it is both theoretically interesting and pragmatically valuable. It is theoretically interesting because studying strategy choices and changes might help cognitive psychologists gain some insights into the underlying mechanisms of human problem solving or other cognitive processes. It is practically meaningful in that a great deal of real-world problem solving involves the problem of choosing a good strategy. In fact, there has already been a large body of research conducted to address the issue of cognitive strategy choices and changes in various domains of cognition, ranging from memory retrieval (Reder, 1987; 1988), through solving some laboratory problems such as number series extrapolation tasks, the Carnival and Missionary problem, and the Tower of Hanoi puzzle (Simon & Reed, 1976; Simon, 1978; Ruiz, 1989), to doing arithmetic calculations (Reder & Ritter, 1990; Siegler & Shrager, 1984; Siegler & Jenkins, 1989), and performing some other more ecologically valid cognitive tasks such as programming (Gray & Anderson,1987; Katz & Anderson, 1988; Pennington, 1987; Soloway, Bonar, & Ehrlich, 1983; Visser, 1987). The research to be reported in this paper is about PASCAL programmer's strategy choices and changes in writing iterative or looping programs.

Three experiments were conducted for this study. The first experiment adopted a simple design in which subjects were free to choose between the two indefinite looping strategies, that is, to use either the "**while-do**" or "**repeat-until**" construct. In the second and third experiments, subjects were either forced or induced to use one of these two looping alternatives. These three experiments were designed to yield some convergent evidence for testing the hypothesis that most PASCAL programmers are adaptable to the nature of the problems while choosing iterative strategies. However, in order to understand what this statement means, we have to discuss first of all the issue of what defines the nature of a problem and what defines a good strategy for that problem. In the following section, we will review some previous research conducted by other investigators on

cognitive strategy choices and changes in the domain of PASCAL iterative programming and we will analyze three types of iterative constructs offered in PASCAL. Following that, the three experiments of this study will be reported and discussed.

COGNITIVE STRATEGY AND ITERATIVE PROGRAMMING

The Study of Cognitive Strategy in Programming

Particularly relevant to our present study is the research done by Soloway and his colleagues (Soloway, et al., 1982; Soloway, 1983). Their study was focused on the "cognitive fit" between the most naturally chosen strategies and the looping constructs provided by the programming language in use. They used a particular problem of iterative programming -- a counting and averaging problem (which was also used as one of the testing problems in this study) -- and found that their subjects overwhelmingly prefer a READ/PROCESS strategy over a PROCESS/READ strategy on that problem. By manipulating different programming languages for subjects to use, their results also indicate that the accuracy of the program to be written is enhanced if it is written in a language which facilitates the preferred strategy for the problem. One of our purposes in this study was to try to replicate the results of the above mentioned study from a different perspective and with more problems. We will contrast our results with theirs in the first experiment to be reported in the next section.

Iterative Strategies in PASCAL Programming

In the domain of computer programming languages, probably PASCAL is the best known conventional programming language in that it is the first advocated and widely accepted structured programming language (Jensen & Wirth, 1974). By now, PASCAL is not only widely used in the computer community but also globally accepted as a tool for

teaching structured programming as a good methodology and style of programming.

With respect to iterative structures, there are three types of iterative or looping constructs furnished by PASCAL -- that is, the **for-to-do**, **while-do** and **repeat-until** statements or constructs. (Henceforth in the paper they will be referred as **F-**, **W-** and **R-** constructs.) The first one is only used in definite looping situations where the number of iterations is known before the iterative part of the program is actually executed. However, the other two may be used in definite as well as indefinite looping cases where the number of iterations may only be dynamically determined by the execution of the looping part. These three types of looping constructs are illustrated by their flowcharts in Figure 1.

Insert Figure 1 about here.

The *difference* between the **F**-construct and the rest is more obvious than the difference between the two indefinite looping constructs. Considering the symmetry of the two indefinite looping constructs, they were chosen to be the subject matter focused on in this research. Note that, for the **W**-construct, the termination condition of iteration is always tested before the execution of the iterative part so that the iteration may be executed zero times (not executed at all). On the other hand, for **R**-construct, the test for termination occurs at the exit of the iterative part so that the iteration will be executed at least once. That is the fundamental difference between these two indefinite looping constructs and in fact is the principle typically taught for choosing between them.

To implement any looping program, either the **W-** or **R**-construct can be selected and applied. However, in certain cases, choosing the **W**-construct would result in a more concise and well-structured program than it would be if **R**-construct is chosen; and in some other cases, it is just the other way around. A general way of converting a **W**-constructed

program to a **R**-constructed, and vice versa, is illustrated in Figure 2 (there are other ways that may be more natural of converting from one construct to the other; for instance, see the examples given in Table 1). From this illustration we can see that if an iterative program is appropriate to be implemented in **W**-construct then it usually would take more statements to be implemented in **R**-construct; and the same is for the case from **R**- to **W**-construct transformation. An exemplar problem which is more suitable to be programmed in **W**-construct is given in Table 1 (Hereafter, this type of problem will be referred as **W**-problems), and another exemplar problem which is more natural to be programmed in **R**-construct is shown in Table 2 (one of **R**-problems). (For the example given in Table 1, instead of following the general way of converting **W**-constructed program to that in **R**-construct as illustrated in Figure 2, we have an **If-then** statement embedded in the **R**-construct.)

Insert Figure 2 about here.

Insert Table 1 about here.

Insert Table 2 about here.

However, as a matter of fact, the situation for choosing between the **W**- and **R**-constructs is sometimes more subtle than it may appear to be at the first blush. For example, the problem shown in Table 3 seems to be neutral with respect to either the **W**-construct or the **R**-construct, because the choice of either **W**- or **R**-construct will only differ in the resulting programs only in the position of the test for iteration termination; hereafter this kind of problem will be referred as **NE**-problems for Neutral Easy. On the other hand,

for the problem shown in Table 4, both **W**- and **R**-constructs seem to be awkward, because it will require either a duplicate of the termination condition test or a duplicate of an action. Hereafter, this kind of problem will be referred as **NH**-problems for Neutral Hard. In fact, the problem given in Table 4 is a typical middle-out looping problem which implies that the most natural iterative construct for implementing it is to have the termination condition test in the middle of the loop, as illustrated by its flowchart in Figure 3. For middle-out looping problems, one way to program them is to use the **goto** statement, as demonstrated in Figure 4. Nevertheless, although that PASCAL does offer the **goto** statement and that sometimes using it wisely does not violate the constraints for a well-structured program, most PASCAL programmers do not like to use the **goto** statement at all. Therefore, in this study we will not consider this option either.[1] However, while planning their programs by drawing flowcharts it is quite possible that programmers use the middle-out looping strategy, corresponding to using the **goto** statement; hereafter, we will denote this strategy as **G**-strategy. The looping strategies corresponding to using the **R**- and **W**-constructs will be denoted as **R**- and **W**-strategies respectively.

Insert Table 3 about here.

Insert Table 4 about here.

Insert Figure 3 about here.

---

[1] Since Dijkstra [1968] pointed out that the **goto** statement is not appropriate for structured programming, whether or not the **goto** statement should be used any further has been a rather controversial issue in programming [Knuth, 1974]. Nowadays, treatments on that topic by typical textbooks on introductory programming in PASCAL vary very widely, ranging from introducing the **goto** statement in the text while at the same time offering advice to use it cautiously [e.g., Findlay & Watt, 1987], through only introducing it as an extra topic in appendix [e.g., Etter, 1988], to "deliberately omitting" any discussion about it [e.g., Martin, 1989].

Insert Figure 4 about here.

EXPERIMENT 1: FREE CHOICES OF ITERATIVE STRATEGIES IN PASCAL PROGRAMMING

As stated earlier, three experiments were conducted for this study. The aim of this first experiment was to see how PASCAL programmers chose different looping strategies on different types of problems in planning and in programming. Therefore, in this experiment subjects were free to choose among the **W**-, **R**-, and **G**-strategies when planning, and between the **W**- and **R**-strategies while programming.

Method

Subjects. The 20 subjects participated in this experiment were from the CMU (Carnegie Mellon University) community; among them 2 were undergraduates, 14 were graduates, and 4 were research assistants. The subjects were reimbursed for participating in the experiment. Although all of them learned PASCAL before taking part in the experience, their experience with it was different from one another, ranging from only having taken an introductory programming course in PASCAL to having done a lot of practical programming experience in PASCAL.

Materials. There were 13 different problems used in this experiment. Among them 4 were classified as **W**-problems, 4 as **R**-problems, 3 as **NE**-problems, and 2 as **NH**-problems (so 5 **N**-problems in total).[2] The first 9 subjects only solved 8 problems -- namely, 3**NE**+2**NH**+2**W**+1**R** (this is the order in which the problems were presented to the subjects); and the rest 11 subjects completed all the 13 problems -- namely,

[2] All these problems can be obtained by writing to the authors.

3NE+2NH+4W+4R (also in the order of presentation).

Procedure. Before the experiment actually began, each subject was asked to fill out a questionnaire form. The form was mainly intended to collect some information about the subjects, such as their positions and affiliations at CMU, their GRE or SAT scores if known, and their self-ratings of programming experience in different languages. For the self-rating of programming experience, the subjects were asked to rate along a scale from zero to five, for each language which they had mastered, according to their own confidence in that language. There were 11 subjects who filled in GRE math scores and 2 subjects who filled in SAT math scores; and the average math scores across both the 11 GRE and 2 SAT scores are 754. Subject's self-ratings of their experience in PASCAL programming range from 2 to 5, and across all the 20 subjects the average is 3.8.

There were two sessions of the experiment for each subject. In the first session, subjects completed either 7 (for those who finished 13 in total) or 4 problems (for those who finished 8 in total); and in the second session, either 6 or 4. In each session, each subject was first asked to draw flowcharts for all the problems to be solved in that session. While drawing flowcharts for the problems, the subjects were explicitly instructed not to think in PASCAL or in any other programming languages. (Most of our subjects had never experienced drawing flowcharts before; however, by looking at a simple flowchart example no one seemed to have difficult in drawing flowcharts.) Only after the subjects finished drawing flowcharts for all the problems did they begin to write PASCAL programs for these problems. All their flowcharts and the first drafts of their PASCAL programs were worked out on paper. When the first draft of the program was finished, the subjects were required to type in the program to the Macintosh II computer and then use the LightSpeed PASCAL version to test it. They had to debug, if needed, the programs until they yielded the correct results which were required by the experimenter; sometimes, the experimenter provided necessary consultation on PASCAL syntax or about the details of the special PASCAL version. The subjects were not allowed to look at their flowcharts while they were

programming; neither were they permitted to access to their solutions on the previous problems when they were on a later problem. While the subjects were doing paper work or working on the computer, they were accompanied by the experimenter; the time they spent on each problem was measured and recorded by the experimenter. For 10 of the 20 subjects, concurrent verbal protocols were taken through the whole experimental sessions. (Among them, 5 finished 8 problems totally; and the other 5 finished 13.)

Results

The three looping strategies -- namely, the **W**-, **R**-, and **G**-strategies -- were all exhibited in subject's flowcharts. On the other hand, while programming in PASCAL, subjects were explicitly advised to only use either the **W**- or **R**-construct; therefore, only two kinds of looping strategies, corresponding to the two looping constructs, are there manifested in their PASCAL programs.[3] Among the 20 subjects who participated in this experiment, 3 subjects overwhelmingly used the **W**-strategy in programming (all of them did 13 problems, and on all the 13 problems they used that strategy), and one subject overused the **R**-strategy (he completed 8 problems; and on 7 of them, including the 2 **W**-problems, he used the **R**-strategy). Here we are only interested in the general pattern of the data from the subjects that varied their use of programming constructs, so these subjects are excluded from the data analysis in this results section. That is, only the data from 16 subjects are analyzed in the following.

Insert Figure 5 about here.

Figure 5 shows the distributions of different strategies chosen by subjects on different categories of problems, classified by their natures, both in planning and in programming.

[3] (Actually, most subjects did not ask whether they were allowed to use the **goto** statement and they did not use it; only very few asked, in which case, they were told by the experimenter to avoid using it.)

Three separate one-way analyses of variance (ANOVAs) were performed on the data for the three different strategies in planning. The statistical results showed that the effects due to the nature of problem were significant, as for **W**-strategy: $F(3, 45) = 4.43$, $p < .01$; for **R**-strategy: $F(3, 45) = 13.06$, $p < 0.001$; and for **G**-strategy: $F(3, 45) = 30.23$, $p < 0.001$. Another one-way ANOVA was performed on the data for the **W**-strategy in programming; the results also revealed significant effects due to the nature of problem, $F(3, 45) = 17.96$, $p < 0.001$. The results seen displayed in the figure are as follows: In planning, the **W**-strategy is a not the preferred strategy for most subjects on any kind of problem; the **R**-strategy is highly chosen on **R-** and **NE-**problems; and the **G**-strategy is favored on **W-** and **NH**-problems. On the other hand, in programming, the W-strategy is dominant for the **W**-problems; the **R**-strategy is dominant for the **R**-problems; however, neither strategy is dominant for the **NE-** or **NH**-problems.

By studying the pattern of the strategies chosen by subjects in the planning phase, we can deduce that what are the natural looping strategies on what types of problems, irrespective to any programming language. Therefore, from the above presented results, it seems that the **G**-strategy is natural on the **W-** and **NH**-problems, while the **R**-strategy is natural on the **R-** and **NE-**problems. On the other hand, by studying the pattern of strategies exhibited in programming, we are indeed confirming the hypothesis that most PASCAL programmers are very sensitive to the nature of the problems being programmed and quite adaptable in choosing corresponding and suitable looping strategies in PASCAL iterative programming.

Discussion

Our results from this experiment are consistent with the conclusions drawn in the study done by Soloway et al (1983). The averaging problem with the sentinel value 99999 used in their study would be classified as a **NH**-problem in our study -- namely, a middle-out looping problem. The data from this experiment indicate that the percentage of

choosing the **G**-strategy on that type of problems in planning is overwhelmingly high. In fact, the **G**-strategy here is what they referred as READ/PROCESS looping strategy in their study, that is, a strategy characterized by the following plan schema:

```
loop
    do begin
        Read (i th value)
        Test (i th value)
        Process (i th value)
    end
```

as explained in Soloway et al. (1983). Their results also indicate that PASCAL does not seem to cognitively facilitate the implementation of this kind of looping strategy; therefore, novice programmers usually have difficulty with this class of problems when facing a choice among looping constructs in PASCAL.

Our results from this experiment seem to demonstrate that most of our subjects are quite adaptable in choosing appropriate looping strategies are quite high. Compared to Soloway et al study, the subjects involved in our study may be relatively more skillful in PASCAL programming. (The subjects in their study were all in an introductory course on PASCAL programming whereas our subjects were generally more advanced.) Let's take a close look at the correlation between ability and adaptability. Subject's experience in PASCAL can be given by two measures -- namely, subject's total problem solving times and their self-ratings of PASCAL knowledge.[4] As the result of the data analysis, we see that subject's self-ratings of their PASCAL programming skill are rather subjective and are unrelated to their problem-solving times (the correlation coefficient is -0.058); therefore, we used the time measure as the more objective one between the two measures. The adaptability is defined as the proportion of the number of the **W**- and **R**-problems on which subject's chosen strategies are consistent with the natures of the problems to the total number of these two classes of problems. The data analysis revealed that the correlation

[4] Due to the fact that 9 subjects in this experiment only finished 8 problems, in the data analysis we only summed over these 8 problems for all the subjects, regardless whether they finished 8 or 13 problems, to calculate their total problem solving times. Also, the total problem solving time does not include the time spent on drawing flowcharts for these problems.

coefficient between the adaptability and self-rating was 0.137, that between the adaptability and time measure is -0.121, and that between adaptability and GRE math scores is -0.147. Thus, we see that subject's expertise in PASCAL programming does not correlate with their adaptabilities of choosing looping strategies very strongly.

EXPERIMENT 2: FORCED CHOICES OF ITERATIVE STRATEGIES IN PASCAL PROGRAMMING

This second experiment was designed to gather further evidence for our conclusion drawn earlier -- that is, most PASCAL programmers are very adaptable in choosing appropriate looping strategies according to the nature of the problems being programmed. Following this, it is reasonable to hypothesize that if PASCAL programmers are forced to use a looping strategy which is incompatible with the nature of the problems for programming then their performance will be hampered in some way. This is essentially the hypothesis that the present experiment was intended to test.

Method

Subjects. As in Experiment 1, all the 24 subjects involved in this experiment were also from the CMU community, among them 14 undergraduates, 8 graduates, and 2 research assistants. The subjects were reimbursed for their participation in the experiment.

Insert Table 5 about here.

Design. There were three conditions in this experiment, with 8 subjects in each condition. The design of the experiment (Table 5) was that subjects were either forced to use the **R-** (Group 1) or **W**-strategy (Group 2) on all the problems or forced to use the

strategies which were incompatible with the natures of the problems -- namely, to use the **W**-strategy on **R**-problems and **R**-strategy on **W**-problems (Group 3). Furthermore, we selected the 8 subjects in Experiment 1 who finished all the 13 problems and for whom we knew their GRE/SAT math scores to make up an additional condition (Group 4), that is, a condition in which subjects were free to choose either the **W**- or **R**-strategy on all kinds of problems. As seen from Table 5, on the **N**-problems the first two groups were forced to use the **R**- and **W**-strategies respectively, whereas the other two groups were free to use either the **R**- or **W**-strategy. According to this design, we would hypothesize that subjects who used the **W**-strategy on **R**-problems would spend more programming time than those who used the **R**-strategy on these **R**-problems, and that subjects who used the **R**-strategy on **W**-problems would spend more time than those who used the **W**-strategy on these **W**-problems. We would also hypothesize that on the N-problems these four groups of subjects would not differ very much in their performance. The 24 subjects in this experiment were randomly assigned to the three original experimental conditions; Table 6 presents the descriptions of these three different groups of subjects, along with the group consisting of the 8 subjects from Experiment 1, of their GRE/SAT quantitative scores and self-ratings of their PASCAL knowledge averaged over the subjects within each group. From the figure we see that subjects in the four conditions are fairly closely matched.

Insert Table 6 about here.

Materials. The same 13 problems as used in Experiment 1were used in this experiment.

Procedure. The same procedure as used in Experiment 1 was followed here. That is, there were two sessions of experiment: in the first session, each subject completed 7 problems; and in the second session, 6 problems. In each session, the subject first did planning then embarked on actual programming. Most of the subjects were accompanied

by the experimenter when they were engaged in experimental sessions, but only for 3 subjects (one in each group) concurrent verbal protocols were collected.

Results

As in Experiment 1, in the planning phase, there was no constraint on how subjects drew their flowcharts; accordingly, we could expect that the pattern of strategies chosen in planning phase would not differ from that obtained in Experiment 1 in any significant sense. In fact, this seems true as we present the results from this experiments in Figure 6 in the same way as we showed the data from Experiment 1 in Figure 5. Here three separate ANOVAs were also performed on the data for the three different types of looping strategies, respectively. The statistics also revealed significant effects due to the nature of problem; that is, for W-strategy, $F(3, 69) = 6.06$, $p < .001$; for R-strategy, $F(3, 69) = 74.79$, $p < .001$; and for G-strategy, $F(3, 69) = 77.74$, $p < 0.001$.

Insert Figure 6 about here.

Insert Figure 7 about here.

To verify the hypothesis mentioned in the design of this experiment, we used the time measure -- namely, the programming time spent by different groups of subjects on different types of problems. The average times of our four groups of subjects are displayed in Figure 7. A two-way ANOVA was performed on the data; the results indicated that the effect due to the nature of problem was statistically significant, $F(2, 14) = 45.29$, $p < .001$; the effect due to the difference among the four groups of subjects was also significant, $F(3, 21) = 23.24$, $p < .001$; and the interaction between them was significant $F(6, 42) = 52.25$, $p < .001$. As seen from the figure, the data indicate that subjects had advantages when they used the strategy

which was compatible with the nature of the problems in a class. More specifically, subjects who used the **W**-strategy on the **R**-problems spent more time than those who used the **R**-strategy on these **R**-problems, and subjects who used the **R**-strategy on the **W**-problems spent more time than those who used the **W**-strategy on these **W**-problems. When subjects were free to choose they were as fast as subjects who were required to use approriate strategy in solving the problems.

EXPERIMENT 3: INDUCED CHOICES OF ITERATIVE STRATEGIES IN PASCAL PROGRAMMING

The purpose of this experiment was also to provide further evidence supporting our general conclusion on PASCAL programmer's adaptability in choosing looping strategies. We were also interested in relating the adaptability with Einstellung effect in problem solving (Luchins & Luchins, 1959). More specifically, the hypothesis being tested in this experiment was that solving a set of **W**- or **R**-problems could have some effect on solving later problems of neutral nature while having less effect on later solving **R** or **W**-problems.

Method

Subjects. As in the previous two experiments the 32 subjects participated in this experiment were either CMU undergraduates (15 of them), graduates (14), or research assistants (3). They were reimbursed for taking part in the experiment. They were from different departments in CMU and had varied experience in PASCAL programming, and they were randomly assigned to the different conditions of the experimental design.

Insert Table 7 about here.

Design. Four conditions were devised in this experiment (accordingly, four groups of subjects), with 8 subjects in each condition. The different conditions were only different from one another in the order in which the testing problems were presented to the subjects (Table 7). The 32 subjects were randomly assigned to these different conditions of the experimental design. Table 8 provides various descriptive statistics for the four different groups of subjects; there is little differences among the groups.

Insert Table 8 about here.

The intention of this experimental design was to see whether there were any inducing or priming effects of having solved a series of problems of one nature upon solving later problems of another type. For instance, for the subjects in Group **W1**, after they had solved a set of **W**-problems, they solved a set of **N**-problems and then a set of **R**-problems. We were interested in whether any inducing effects would be exhibited on switching from **W**-problems to **N**- and **R**-problems.

Materials. The 13 problems used in this experiment were the same as those used in the previous two experiments. The sequences in which they were presented to subjects were different according to the designated conditions.

Procedure. As in the previous two experiments, before the experiment actually began, each subject was required to fill out a questionnaire form. However, unlike the previous experiments, subjects were only required to work out flowcharts and PASCAL programs on paper in this experiment. The correctness of their programs was judged by the experimenter; if their programs were incorrect then they had to revise them. This procedural change from the previous experiments to the present one was because we here were mainly interested in the looping structures of the programs produced by subjects, not very much in timing subject's problem solving processes. Consequently, although subjects

in this experiment were required to record down the time they spent on each problem, the data of their programming time spent on different types of problems were not as accurate as what we had in the previous two experiments. [5]

Results

Again, as in Experiment 1, there were 5 subjects (5/32= 15.6% of the total subject population of this experiment) who idiosyncratically overused either the **W**- or **R**-strategy on all the problems. Among these subjects, two used the **W**-strategy on all 13 problems; one used the **R**-strategy on all the problems; and the other two used the **R**-strategy on 12 problems, including on four **W**-problems. These subjects were excluded from the following data analysis. The data from the experiment expressed as the percentage usage of the **W**-strategy on the three different types of problems by four groups of subjects are shown in Figure 8. A two-way ANOVA was performed on the data; the results indicated that

---

Insert Figure 8 about here.

---

the effect due to the nature of problem was statistically significant, $F(2, 14) = 170.92$, $p < .001$; the effect due to the different presentation order as for different groups of subjects was marginally significant, $F(3, 21) = 4.20$, $p < .02$; and the interaction between them was significant $F(6, 42) = 4.62$, $p < .001$. As seen from Figure 8, the pattern of the data seems to conform to the hypothesis being tested. That is, on the **N**-problems, subjects from Group **W1**, who had used the **W**-strategy on the previous set of **W**-problems, tended to use that strategy more often than those from Group **R1**. The same data pattern holds for Group **W** and Group **R** in that the immediately preceding problems biased the strategies they chose

[5] In this experiment, all the subjects were explicitly instructed to use only the **W**- and **R**-constructs. However, one subject, a research assistant, did not follow the instruction very faithfully and on many problems used the **goto** statement; this subject was subsequently replaced by an additional subject recruited. The information this subject stated on the questionnaire revealed that he was a novice programmer in general; and many of his programs were badly structured.

on the **N**-problems, although the disparity between these two groups of subjects on the **N**-problems is less smaller than that between Group **W1** and Group **R1**. Furthermore, from the data we also can see that there are no significant effects of having previously exposed to a set of **R**-problems upon solving a set of **W**-problems later, and vice versa.

CONCLUSIONS

In generalizing our conclusion to more general population of PASCAL programmers, we are aware of the fact that all the subjects involved in this study were from CMU and that CMU is a relatively highly computer-oriented educational institution among all the universities and colleges in the world. The typical introductory textbook on PASCAL programming used at CMU is Miller and Miller (1986) in which both the **while-do** and **repeat-until** statements are introduced (the former introduced first) and the principle for judging when to use which is also introduced. Therefore, for those CMU undergraduates who studied PASCAL at CMU, it is plausible to relate their adaptability of choosing appropriate looping constructs to their PASCAL learning experience at CMU.

On the other hand, the failure to find a relationship between programming ability and tendency to be adaptive suggests that our results may not be restricted to high ability subjects. [6] With some uncertainty about the range of subjects our conclusions may apply to, we feel that these three experiments have established the following conclusions about strategies for iterative programming in PASCAL:

1. Independent of any particular programming language, there are two types of looping strategies most frequently and naturally chosen by people while planning iterative solutions to the problems being solved. People usually choose the **R**-strategy if it seems easier to use it, or choose the **G**-strategy otherwise.

2. In PASCAL programming, most programmers seem to be very sensitive and

[6] We should note that in each experiment we found a minority of subjects who just used one programming construct; for these subjects the issue of strategy selection does not arise.

adaptable to the nature of the program. They would choose between the **W-** and **R-** strategies according to which will produce a concise and well structured program.

3. Adaptability in choosing programming strategy does not seem to be related to the programmer's ability.

4. When subjects are forced to use a non-preferred strategy their programming suffers.

5. When the problem does not have a preferred strategy, subjects show tendency to continue to use the last strategy they used.

The high degree of sensitivity displayed by our subjects to problem characteristics is consistent with other results from other domains in cognitive psychology (e.g., Reder 1987, 1988; Reder & Ritter, 1990; Siegler & Shrager, 1984) indicating that subjects are highly adaptive in their strategy choices. Reder (1987, 1988) showed in a different domain, question-answering, that subject's strategy selection was jointly influenced by the nature of the problem (question) and by what strategies have recently been used. Our results showed both effects; however, unlike her, we see that problem nature totally dominates past experience. This may reflect the fact that the problem nature manipulation in our experiments was stronger than her manipulation which was recency of information. In more recent research on arithmetic problem solving, Reder and Ritter (1990) have also found dominant effects of problem type. Our research is also consistent with the emphasis of Siegler and Jenkins (1989) on the highly adaptive nature of strategy selection. Therefore, the pattern of strategy selection seen in these experiments in consistent with results based on very different populations working in very different domains.

REFERENCES

Dijkstra, E. (1968). GO TO statement considered harmful. *Communication of ACM.* Vol. 11, No. 3 (November, 1968).

Etter, D. M. (1988). *Problem Solving in PASCAL -- for Engineers and Scientists.* Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc.

Findlay, W. & Watt, D. A. (1987). *PASCAL: An Introduction to Methodical Programming.* (3rd edition). Rockvill, MA: Computer Science Press.

Gray, W. D. & Anderson, J. R. (1987). Change-episodes in coding: when and how do programmers change their code? in Olson, G. M., et al. (eds), *Empirical Studies of Programmers: Second Workshop.* New York: Ablex Publishing Corp.

Jensen, K. & Wirth, N. (1974). *PASCAL User Manual and Report.* New York: Springer-Berlag.

Katz, I. R. & Anderson, J. R. (1988). Debugging: an analysis of bug-location strategies. *Human-Computer Interaction*, Vol. 3, pp. 351-399.

Knuth, D. (1974). Structured programming with GO TO statements. *ACM Computing Surveys.* Vol. 6, No. 4. Reprinted in Yeh, R. (1977) (ed.), *Current Trends in Programming Methodology.* Englewood Cliffs, NJ: Prentice-Hall.

Luchins, A. S., & Luchins, E. H. (1959). *Rigidity of Behavior: A Variational Approach to the Effects of Einstellung.* Eugene, OR: University of Oregon Books.

Martins, J. P. (1989). *Introduction to Computer Science Using PASCAL.* Belmont, CA: Wadsworth Publishing Company.

Miller, P. L., & Miller, L. W. (1986). *Programming by Design: A First Course in Structured Programming.* Pittsburgh, PA: Carnegie Publishing, Inc.

Pennington, N. (1987). Comprehension strategies in Programming. in Olson, G. M., et al (Eds), *Empirical Studies of Programmers: Second Workshop*, Ablex Publishing Corp.: New York.

Reder, L. M. (1987). Strategy selection in question answering. *Cognitive Psychology*, 19(1), 90-138.

Reder, L. M. (1988). Strategic Control of Retrieval Strategies. In G. Bower (Ed.), *The Psychology of Learning and Motivation*, Vol. 21, New York: Academic Press.

Reder, L. M. & Ritter, F. (1990). The effect of feature frequency on feeling of knowing and strategy selection for arithmetic problems. (to be submitted)

Ruiz, D. & Newell, A. (1989). Tower-noticing triggers strategy-change in the Tower of Hanoi: a Soar model. *The Proceedings of the Eleventh Annual Conference of the*

Cognitive Science Society. Ann Arbor, Michigan.

Siegler, R. S. & Jenkins, E. A. (1989). *How Children Discover New Strategies*. Hillsdale, NJ: Erlbaum.

Siegler, R. S. & Shrager, J. (1984). Strategy Choices in addition and subtraction: How do children know what to do? In C. Sophian (Ed.), *Origins of Cognitive Skills*. Hillsdale, NJ: Erlbaum.

Simon, H. A. (1978). What the knower knows: Alternative strategies for problem solving tasks. In Klix, F. (ed.), *Human and Artificial Intelligence*. Berlin: VEB Deutscher verlag der Wissenschaften.

Simon, H. A. & Reed, S. K. (1976). Modeling strategy shifts in a problem-solving task. *Cognitive Psychology*, Vol. 8, pp. 86-97.

Soloway, E.(1986). Learning to program = learning to construct mechanisms and explanations. *Communication of ACM*. Vol. 29, No. 9 (September, 1986).

Soloway, E., Bonar, J., & Ehriich, K.(1982). What do novices know about programming? in Shneiderman, B. & Badre, A. (Eds), *Directions in Human-Computer Interactions*, New York: Ablex Publishing Co.

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: an empirical study. *Communication of ACM*. Vol. 26, No. 11 (November, 1983).

Soloway, E. & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*. Vol. SE-10, No. 5 (September, 1984).

TABLE 1.

An examplar W-problem which is more naturally solved by using W looping construct.

Problem statement

Write a program which copies a part of an input file into an output file. The input file is structured as follows:

Name-1	Integer-1
Name-2	Integer-2
---	---
Name-n	Integer-n
'END'	Integer-(n+1)
---	---

Copy the file until the 'END' is reached (NOT including the item 'END').

A model W-solution

```
program ExampleProg1.1
  type
    Entry = record
       Name: packed array[1..3] of char;
      Number: integer
    end;
  var
    InputFile, OutputFile: file of Entry;
   Temp: Entry;
 begin
     Reset(InputFile);
     Rewrite(OutputFile);
      while InputFile^.Name<>'END' do
       begin
            Read(InputFile, Temp);
            Write(OutputFile, Temp)
       end
 end.
```

A model R-solution

```
program ExampleProg1.2
  type
    Entry = record
       Name: packed array[1..3] of char;
      Number: integer
    end;
  var
    InputFile, OutputFile: file of Entry;
   Temp: Entry;
 begin
     Reset(Inputfile);
     Rewrite(Outputfile);
    repeat
        Read(InputFile, Temp);
        if Temp.Name<>'END' then
       Write(OutputFile, Temp)
    until  Temp.Name='END'
 end.
```

TABLE 2.

An examplar R-problem which is more naturally solved by using R looping construct.

Problem statement

Write a program which copies a part of an input file into an output file. The input file is structured as follows:

Name-1	Integer-1
Name-2	Integer-2
---	---
Name-n	Integer-n
'SUM'	Integer-(n+1)
---	---

Copy the file until the 'SUM' is reached (including the item 'SUM').

A model R-solution

```
program ExampleProg2.1
  type
    Entry = record
        Name: packed array[1..3] of char;
       Number: integer
    end;
  var
    InputFile, OutputFile: file of Entry;
   Temp: Entry;
 begin
     Reset(InputFile);
     Rewrite(OutputFile);
    repeat
        Read(InputFile, Temp);
        Write(OutputFile, Temp)
     until  Temp.Name ='SUM'
 end.
```

A model W-solution

```
program ExampleProg2.2
  type
    Entry = record
        Name: packed array[1..3] of char;
       Number: integer
    end;
  var
    InputFile, OutputFile: file of Entry;
   Temp: Entry;
 begin
     Reset(Inputfile);
     Rewrite(Outputfile);
     Temp.Name:='NOT';
      while Temp.Name<>'SUM' do
       begin
          Read(InputFile, Temp);
          Write(OutputFile, Temp)
       end
 end.
```

TABLE 3.

An examplar NE-problem which is neutral with respect to either the W or R looping construct, and can be easily solved in either construct.

Problem statement

Write a program to read in a series of integers until their sum is greater than 10000 (It is assumed that each integer in the series is less than 10000). This program should also calculate the average of the integers which have been read in.

A model W-solution

```
program ExampleProg3.1
  var
      Number, Sum, Count: integer;
  begin
     Sum:=0;
     Count:=0;
      while Sum<=10000 do
        begin
            Read(Number);
            Sum:=Sum+Number;
            Count:=Count+1
        end;
       writeln('Sum=', Sum);
       writeln('Average=', Sum/Count)
  end.
```

A model R-solution

```
program ExampleProg3.2
  var
      Number, Sum, Count: integer;
  begin
     Sum:=0;
     Count:=0;
    repeat
        Read(Number);
        Sum:=Sum+Number;
        Count:=Count+1
    until  Sum>10000;
     writeln('Sum=', Sum);
     writeln('Average=', Sum/Count)
  end.
```

TABLE 4.

An examplar NH-problem which is neutral with respect to either the W or R looping construct, but hard to be solved in either.

Problem statement

Write a program which interactively reads in a month number. That is, the program is expected to get a number in the ranger of 1 through 12. If the value obtained from the input is out of the range, then an appropriate message should be displayed and an additional value should be inputted. This process goes on until a correct month number is obtained, then the correct number is to be displayed.

A model W-solution

```
program ExampleProg4.1
  var
     Month: integer;
  begin
     Write('Input a month number:');
     Readln(Month);
      while (Month<1) or (Month>12) do
        begin
            Write('Bad input, try again:');
           Readln(Month)
        end;
       Writeln('Correct input:', Month)
  end.
```

A model R-solution

```
program ExampleProg4.2
  var
     Month: integer;
  begin
     Write('Input a month number:');
    repeat
        Readln(Month);
         if not Month in [1..12] then
         Write('Bad input, try again:');
     until  Month in [1..12];
     Writeln('Correct input:', Month)
  end.
```

TABLE 5.

The design of Experiment 2

on forcing subjects to use predetermined W- or R- looping strategies.

Problems	The sequence of the problems		
	5 N-problems	4 W-problems	4 R-problems
Conditions	Looping constructs forced to use in the experiment		
Group 1	**R**	**R**	**R**
Group 2	**W**	**W**	**W**
Group 3	**Free**	**R**	**W**
Group 4	**Free**	**Free**	**Free**
(Group 4 consists of 8 subjects from Experiment 1)			

TABLE 6.

The differences among four groups of subjects involved in Experiment 2.

Conditions	No. of Undergraduate Subjects	No. of Graduate Subjects	No. of Res. Assitant Subjects	Averaged GRE/SAT Quan. Scores	Self-ratings of PASCAL knowledge
Group 1	3	3	2	708.0	4.25
Group 2	5	3	0	720.0	4.13
Group 3	6	2	0	723.8	4.00
Group 4	2	5	1	743.8	3.88

TABLE 7.

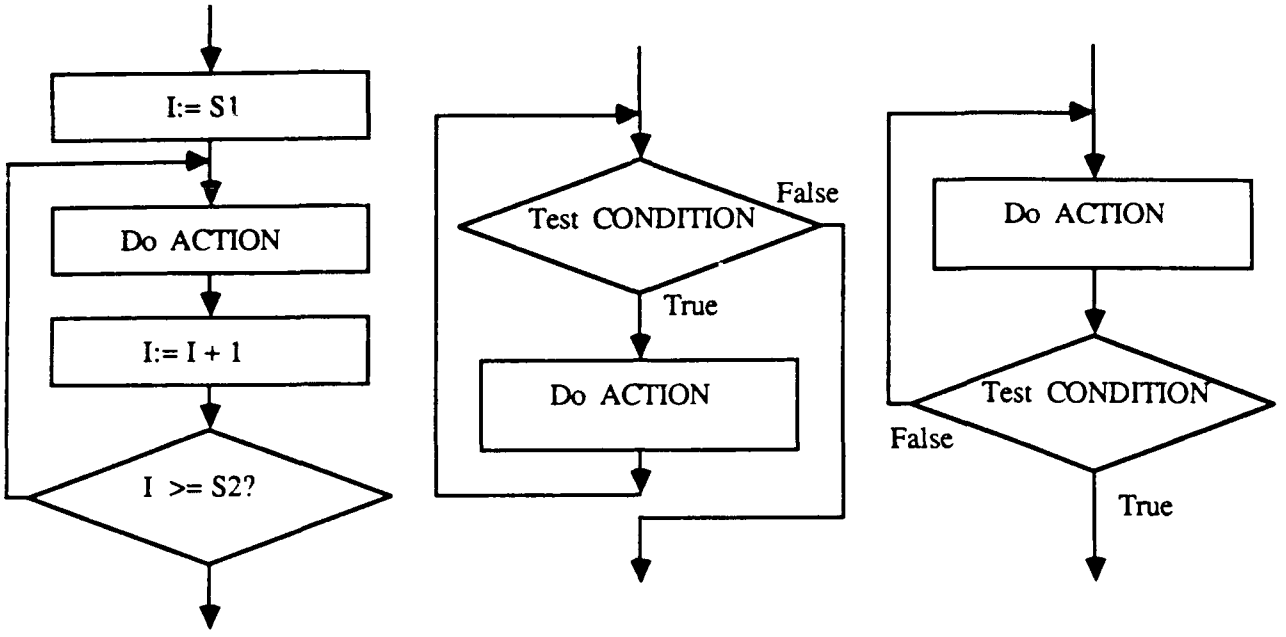
The design of Experiment 3 on inducing subjects to use either the W or R looping construct.

Conditions	The sequence of the problems to be presented to subjects														
Group W1	W	W	W	W	N	N	N	N	N	R	R	R	R		
Group R1	R	R	R	R	N	N	N	N	N	W	W	W	W		
Group W	W	W	W	W	R	R	R	R		N	N	N	N	N	
Group R	R	R	R	R	W	W	W	W		N	N	N	N	N	

TABLE 8.

The differences among four groups of subjects involved in Experiment 3.

Conditions	No. of Undergraduate Subjects	No. of Graduate Subjects	No. of Res. Assitant Subjects	Averaged GRE/SAT Quan. Scores	Self-ratings of PASCAL knowledge
Group W1	4	4	0	712.5	3.25
Group R1	3	4	1	756.0	3.88
Group W	4	3	1	738.0	3.88
Group R	4	3	1	734.1	3.88



Definite looping
for I= S1 to S2
do ACTION
(F-construct)

Indefinite looping

while CONDITION
do ACTION
(W-construct)

repeat ACTION
until CONDITION
(R-construct)

FIGURE 1. Three types of iterative constructs facilitated by PASCAL programming language.

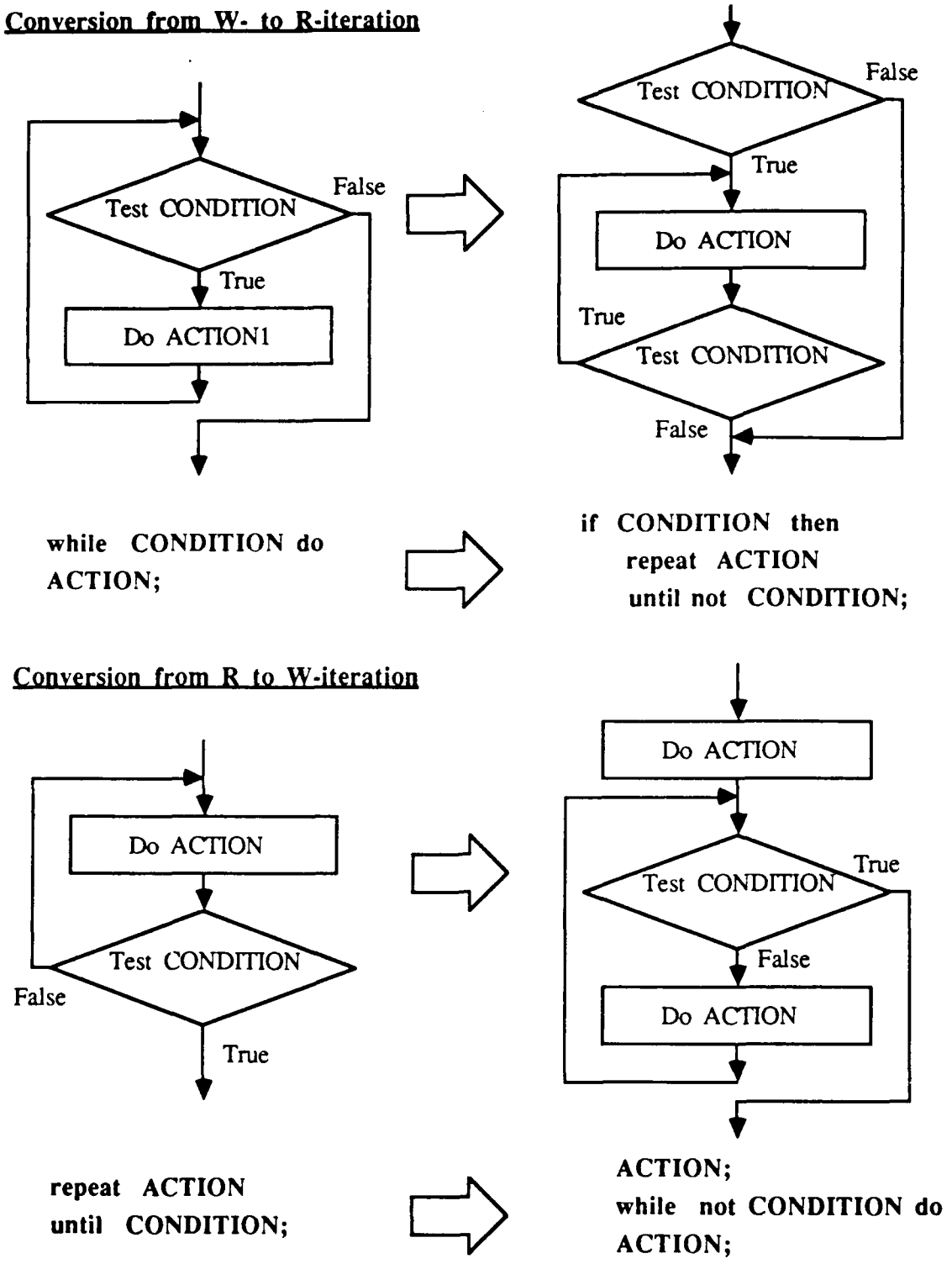


FIGURE 2. The transformation between a W-iterative program and a R-iterative program.

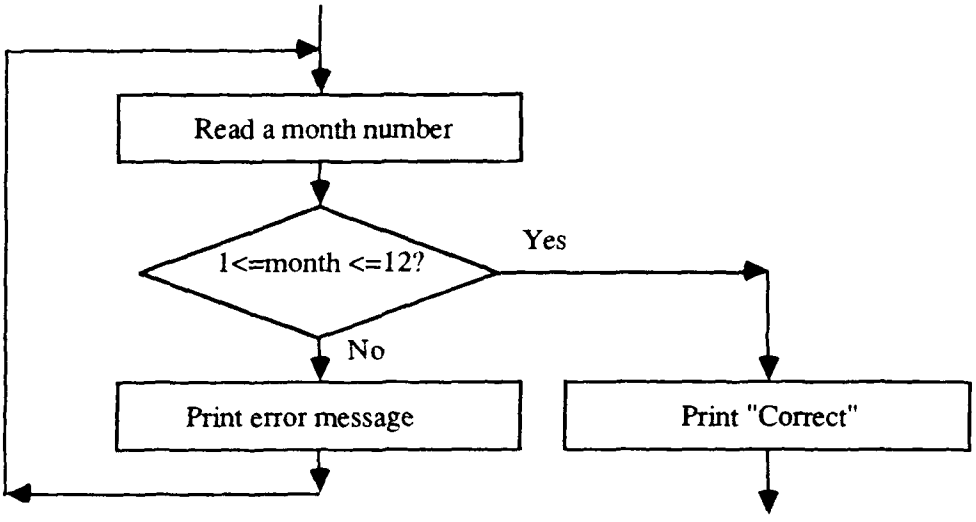
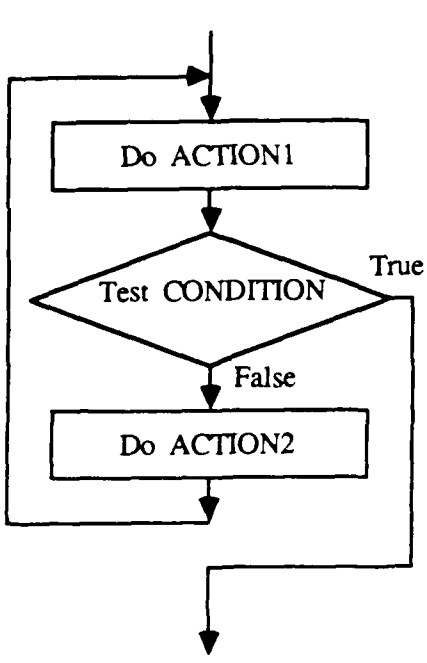
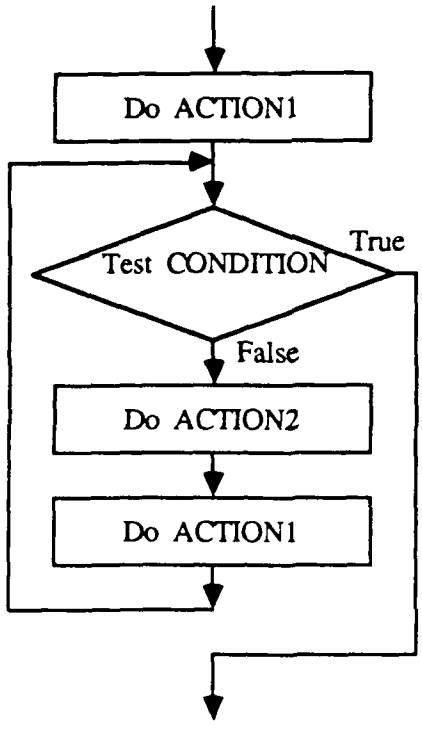
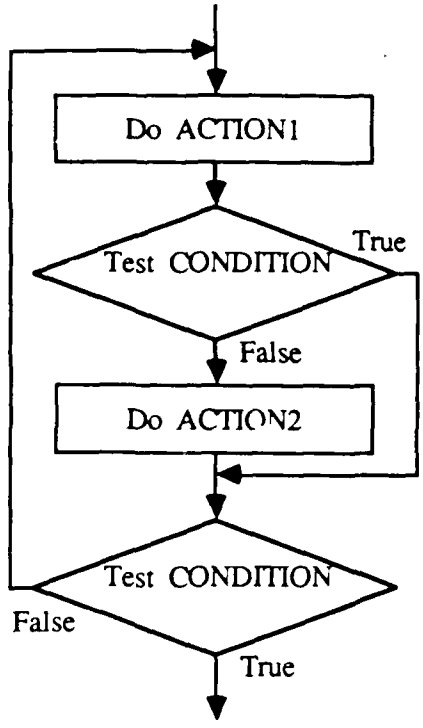


FIGURE 3. A model flowchart for the examplar NH-problem shown in Table 4 -- a typical middle-out looping structure.



Middle-out looping





Its implementation by using "goto" statements:

```
L1: ACTION1;
    if CONDITION
       then goto L2;
    ACTION2;
    goto L1;
L2:
```

(G strategy)

Its implementation in "W" construct:

```
ACTION1:
while CONDITION do
   begin
      ACTION2;
      ACTION1
   end;
```

(W strategy)

Its implementation in "R" construct:

```
repeat
    ACTION1;
    if not CONDI.ION
    then ACTION2;
until CONDITION;
```

(G strategy)

FIGURE 4. Middle-out looping and its implemention by using either the W or R construct.

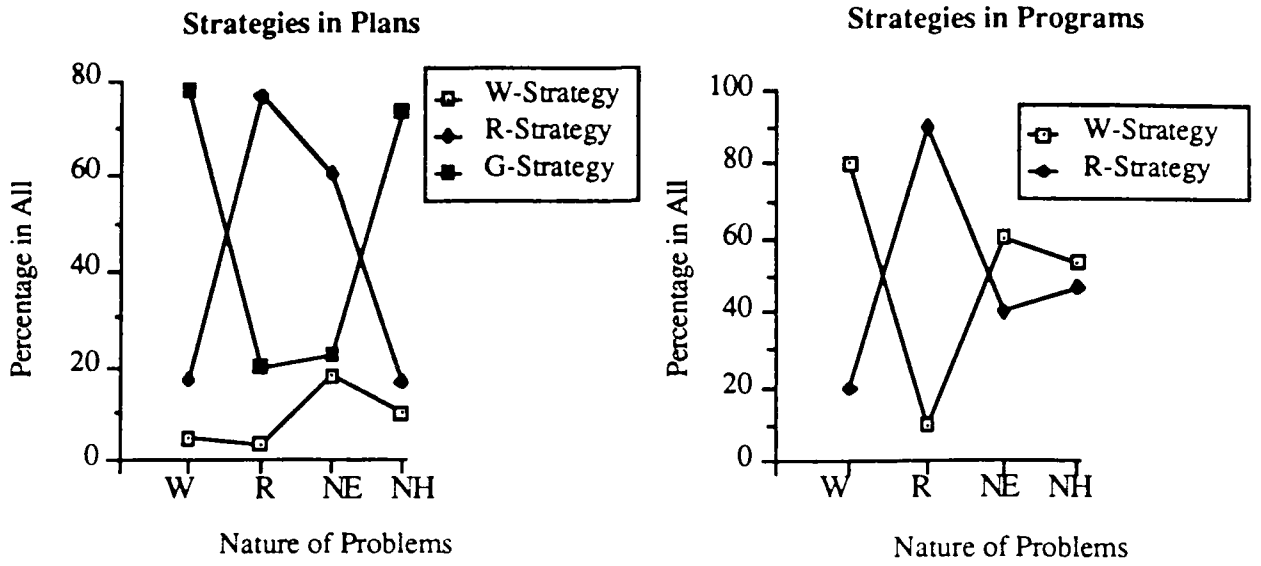


FIGURE 5. The structures (or strategies) manifested in the subject's plans and programs in Experiment 1.

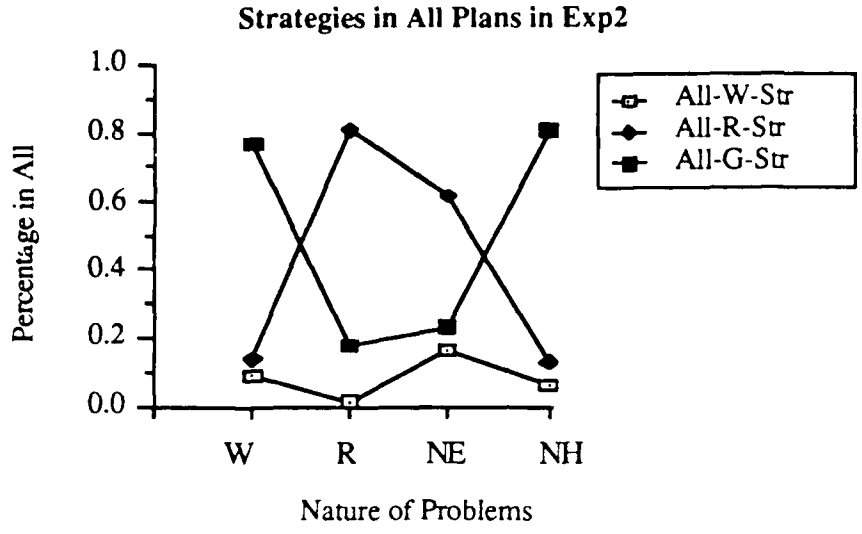


FIGURE 6. The looping strategies manifested in the plans written by the subjects from the three groups in Experiment 2.

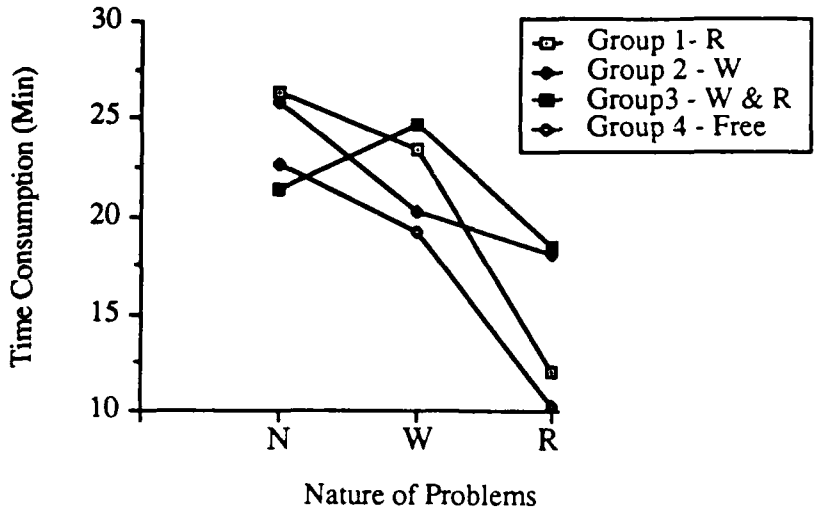
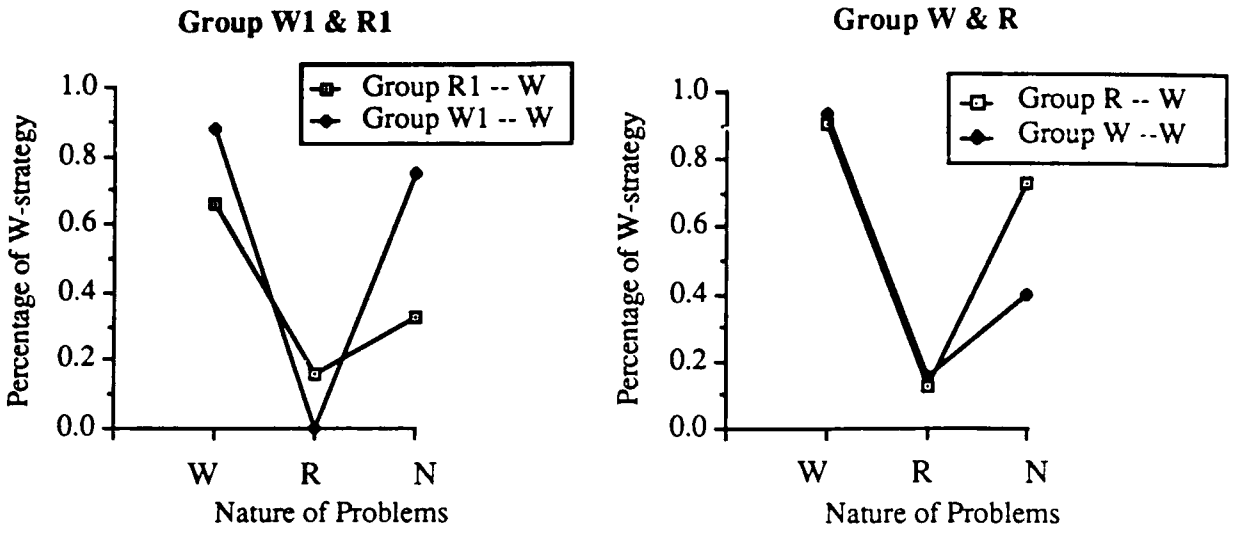


FIGURE 7. The time spent by the subjects in Experiment 2 while they were forced to use predetermined looping constructs with a comparison to the time spent by 8 subjects in Experiment 1 (Group 4) while they were free to choose looping constructs.



NOTE: The problems are not listed in the order of presentation in the experiment.

FIGURE 8. The structures (or strategies) manifested in the subject's programs in Experiment 3 while they were induced to use particluar types of looping constructs.