2) If the error cannot be located exactly in 1) then tracing through the program by hand or by inserting printouts in the approximate area will be necessary.

2. How can the error be corrected?

a. What exactly is the error?

1) State what the error is.

2) State what type of error it is, e.g., logic.

b. What can be done to correct the error?

1) If it is a syntactic or semantic error, change the statement(s) so that they satisfy the rules of the language.

2) If it is pragmatic, alter the statement(s) so that they reflect the meaning of the language structures defined for the language.

3) If it is a logic error, reevaluate the work that has been done, starting with Problem Analysis. Work through all steps including Program Coding and Program Testing, being careful to correct all phases of the work besides the program code.

Step 9 - Documentation

The Documentation step is concerned with the production of written records which give information about the program and its relationship to the surrounding environment. This information may be of several different types depending upon the requirements of the users of the documentation. Documentation is especially important in the commercial

environment where it includes such items as feasibility studies, system design, detailed program logic, operation procedures for running the program, user instructions, and management aids [Gray and London, 1969, pp. 14-16]. This extensive documentation is used for communication not only between the producer of the program and the user(s) but also between the programmers who write the program. Because of the importance of this kind of documentation, standards of various types have been devised in an effort to provide consistent guides for its production [Gray and London, 1969]. Even when the program is intended for the personal use of the programmer, certain documentation is a wise investment for use both during program development, e.g., that which is produced during each step in the programming process and used in subsequent steps, and afterward during program utilization; such documentation is the objective of this step in the transformational process.

Documentation as a step in the transformational process is not accomplished all at once after the program is completed. Certain documentation information should be available from work done throughout the programming process. Once the program is completed a summary of that work plus a description of the completed program should be made which can be used both as an aid to using the program and as a basis for program modification, if it is necessary. The summary documentation should include the following basic information:

1) Program Definition Description which describes the purpose of the program, the inputs to the program and how to prepare them, and the outputs from the program and how they appear. This should include all the information needed to use the program.

2) Program Logic Description which details the logic of the program. This may be in the form of a systems flowchart, a more detailed program flowchart and/or decision tables along with a narrative explanation of the following:

   a) the purpose of each variable used

   b) the purpose of each subprogram along with its inputs and its outputs

   This should include enough information so that in case of errors, it can be used for debugging purposes, and in case of necessary modifications, it can be used for updating the program.

3) Program Listing which gives the entire program. Included in this should be some commentary which gives a brief summary of each part of the program as to its purpose.

The Documentation step takes information from other steps in the transformational process. It starts with the information about the purpose of the program and what its inputs and outputs are — this from the Problem Definition step. It also takes the information about the explicit form of the inputs and outputs; this may come from either the Problem Definition step or the Solution Plan step. Detailed flowchart and narrative information about the program comes from the Solution Plan and Algorithmic Description steps which has been updated to reflect the completed program. Of course, the program listing comes from the completed Program Coding, Program Testing, and Debugging steps. The commentary in the listing itself may be inserted during Program Coding or it may be inserted during this Documentation step.

Specifications for Documentation

1. What information is necessary so that the program can be used?
   a. List the purpose of the program.
   b. List the inputs to the program along with
      1) the input media
      2) the field specifications for each input
      3) the relationships among the inputs, i.e., which ones are dependent upon others and what is their relative ordering
   c. List the outputs from the program along with
      1) the output media
      2) the form of each output
   d. List any options which may be used, e.g., suppression of certain outputs, etc.
   e. List any other auxiliary information about using the program on the computer system, e.g., field length, time limit bounds, etc.
2. What information is necessary for use either in debugging or modifying the program at some future time?
   a. Describe the overall program logic by giving
      1) the relationships among the main program and subprograms
      2) identifying the purpose of each subprogram
      3) identifying the inputs to and outputs from each subprogram

This may be in the form of a systems flowchart with an accompanying verbal description

b. Describe the details of the program by giving

1) the inner workings of the main and subprograms

2) identifying each variable used, both as to its purpose and its data structure and values

This may be in the form of a program flowchart or decision tables with an accompanying verbal description.

3. What information is needed to provide the total program description?

a. Provide a complete program listing

b. Include brief comments in the listing which identify the following:

1) Purpose of each subprogram or other logical segment

2) Purpose of each variable

Step 10 - Re-analysis

The Re-analysis step in the programming process is not really a step which contributes to the production of the program which has been completed; however, this step is included in the transformational process since it contributes to the overall development of the programmer and his skills. By looking back over not only the completed program and its documentation but also over the methods used in developing the program, the programmer may observe new problem solving methods, new program design methods, and new language structure construction methods which he can consciously add to his repertoire of programming tools. He may also

wish to pursue some tangential ideas or try to clarify some uncertainties about a programming problem which he programmed around at the time. He may question himself about how the new ideas he has observed or used during this last programming task can be utilized on other problems, and he may also ask himself what things he would do differently in the task he has just completed if he had to do it all over again. Such re-analysis is, of course, applicable to any problem solving task [Polya, 1957, pp. 14-16, 61-64]. It can be especially useful in programming where a better way to do some task may result in less time being spent on producing a solution plan or in a shorter solution plan, both of which are desirable goals.

The goal, then, of the Re-analysis step is to evaluate the program and the methods used to produce it in an effort to increase programming skills. Two levels of perspective are needed here. First, a close view of the products of each step in the process and the program itself is useful in establishing why certain aspects of the products and the program were constructed the way they were. Second, a broader view of the entire process used in producing the program is useful in establishing why certain methods were used instead of others. Therefore, the Re-analysis step needs the products of all the steps in the transformational process as a basis for evaluation. The product of this final step in the transformational process is a summary of new techniques to be added to the repertoire of the programmer; it may also contain modifications to already established techniques as well.

Specifications for Re-analysis

1. What new aspects of language structure construction have become apparent during the programming process?
   a. List any new language structure used in the program which has not been used previously giving
      1) the reasons why it was used
      2) the applications for which it seems potentially useful
      3) its advantages over other language structures
      4) its disadvantages
   b. List any language structure which was avoided (or was programmed around) giving
      1) the reason why it was avoided
      2) the result of attempts to clarify its use, either as a result of consulting someone else or the literature or as a result of special programming tests using that language structure
   c. List any language structure which, while not being available at present, would be convenient to have, giving
      1) the reason why it would be useful
      2) a possible way to implement it
2. What new aspects of program design have become apparent during the programming process?
   a. List any new programming concept which was used for the first time (this may be a type of data structure, a new way of organizing subprograms, etc.) giving

1) the reasons why it was used

2) the applications for which it seems particulary useful

3) its advantages

4) its disadvantages

b. List any new programming concept which, although it was not used, was considered as being a potentially useful one, giving

1) the reason why it would be useful

2) its advantages over presently used programming concepts

3) its disadvantages

3. What new aspects of problem solving, especially useful for programming, have become apparent during the programming process?

a. List any new approach consciously used in the programming task (e.g., a process described in the literature such as "step-wise refinement") giving

1) how satisfactorily it worked

2) its advantages

3) its disadvantages

b. Looking over the entire process, describe the kinds of problem solving processes which were used (e.g., are search paths depth first or breadth first in establishing solutions?) and how satisfactorily they worked. How could they be improved?

c. Describe how the given problem could have been approached differently and how the subsequent program could have been alternatively produced considering the results of this re-analysis.

CHAPTER 3

APPLICATION OF THE METHOD TO AN INTRODUCTORY COURSE

3.1 Introduction

This section describes how the proposed method for designing programs can be incorporated into an introductory computer science course. The incorporation of this method takes place in two different contexts. The first context is that of including the method in the content of the course. This is, of course, the most obvious. The second context is that of using the method itself as the basis for the entire course structure; that is, all course content is presented within the framework of the proposed method. This latter use of the method alters the orientation of the course from that which normally exists. Before proceeding to the specific aspects of utilizing the method, a brief description of the course is given.

3.2 The Introductory Course

The introductory computer science course used for the basis of further discussion is patterned after Course B1 — Introduction to Computing as described in "Curriculum 68" [ACM, 1968, p. 156]. The course is normally encountered at the freshman level and it is the prerequisite for all subsequent computer science courses. Students taking the course should have had three units of high school mathematics (recommended by "Curriculum 68") or at least high school algebra.

Course objectives

The purpose of this course, as stated in "Curriculum 68," is to

"provide the student with the basic knowledge and experience necessary to use computers effectively in the solution of problems" [ACM, 1968, p. 156]. A student finishing the course, in light of the above goal, should be able to meet the following objectives:

1) He should be able to write computer programs of moderate difficulty, in at least one higher level language, which are
   a. solutions to given programming problems
   b. correct, i.e., produce correct output
   c. well-designed
   d. tested
   e. debugged
   f. documented
2) He should be able to analyze programs well enough to
   a. explain the mechanics of the program, i.e., trace through the statements, giving values generated, paths followed, etc.
   b. state if it is correct, how efficient it is, how well organized it is, and how it can be improved.
3) He should be able to discuss the various aspects of the programming task, especially the steps involved in the task. He should also be able to discuss how the steps relate to each other and to the design of the program and how the program characteristics relate to program design.
4) He should be able to discuss
   a. the organization of a computer and of a computer system
   b. the general history of computers
   c. some applications of computers

Course content

If the student is to meet these objectives, he must be presented adequate course content and he must gain experience in working with the information presented. The course content required to support the objectives can be classified roughly into four categories. These are

1) background information — this includes the historical background of computers, the organization of computers and computer systems, the applications of computers, and any other information needed for the student to interact with the specific system being used
2) programming concepts — this includes those concepts commonly used in the programming task which transcend the specific language being used; for example, the concepts of data, instruction, and iteration are applicable to the programming task whether LISP, COBOL, or MIX is used
3) language concepts — this includes all elements of the specific language being used and how the elements can be legally combined to produce concrete applications of the programming concepts
4) a method for designing programs — this is a method which encompasses the entire programming task and which incorporates general problem solving techniques in the designing of programs

The course content outlined in "Curriculum 68" is adequate for the first three categories listed; however, no method for designing programs is suggested. The proposed method for designing programs is, therefore,

to be included in the course content outlined in "Curriculum 68" to give the introductory course a content which will support the objectives.

In order for the student to gain a working knowledge of the course content, he needs practice, especially on those aspects which require working skills on his part — namely, writing and analyzing programs. Practice will be, then, of two forms. There will be exercises over new concepts. There will also be the application of the method for designing programs to various programming assignments. Specific discussion is given later concerning these exercises and assignments.

Course structure

The course structure commonly used for introductory courses is centered around the presentation of a new programming concept and its transformation into a concrete language representation. For example, if the new concept of iteration is introduced then emphasis is typically placed on the construction of iterative structures in the language being used. Naturally this is an important part of the programming task; however, it is also important to establish how the new concept fits into the programming task as a whole. Questions such as the following should not be overlooked:

1) what characteristics in a given problem may indicate the use of this concept?
2) what characteristics in a solution plan may require the use of this concept?
3) what ways can this concept be represented algorithmically?
4) what ways can this concept be represented in the language being used?

5) what special characteristics does this concept possess which should be tested for?

6) what are common errors when this concept is used in a program?

7) what characteristics will the program possess as a result of this concept?

If the student is to learn to construct programs which are well-designed, he must see how new concepts fit into the overall programming task and how they affect the characteristics of the resulting program. Therefore, the course structure of the introductory course is based on the proposed method for designing programs so that, rather than emphasizing the language representation of specific programming concepts, the emphasis is on the entire programming task, of which the language representation of specific programming concepts is one of several aspects.

The advantages of using the proposed method as the framework of the course structure are twofold. First, it broadens the emphasis of the course to include the entire programming task. Secondly, while the student is exposed to the proposed method explicitly as part of the course content, he is also exposed to it implicitly as new concepts are related to all parts of the method. Thus, repetition of the method occurs throughout the course as new material is presented, giving the student the opportunity to see the application of the method under various conditions.

3.3 Presenting the Method for Designing Programs to the Student

The proposed method for designing programs should be introduced to the student as early as possible in the course so that it can be used

as the framework within which the core of the course content is presented. Initial presentation of the method will necessarily be at an abstract level where an overview of the entire method is presented along with an outline of its ten steps. This establishes the framework of the method. Details are filled into this framework through the introduction of new concepts as the course progresses until at the end of the course the student will have acquired a working knowledge of the proposed method for designing programs in tandem with the necessary programming and language concepts.

The discussion below gives one approach to the presentation of the method to the student. It suggests an organization for the course and an order for content presentation along with coordinated exercises and assignments.

One important aspect of this approach is that the ordering of concepts differs significantly from that found in a typical course. Because the proposed method for designing programs is concerned with the characteristics of a well-designed program, program structure is a central concept. Subprograms are an integral part of program structure and are directly related to the subproblems of any given problem. In order to take advantage of the overall unity found in the proposed method, especially with respect to the relationship between parts of a given problem (subproblems) and parts of the corresponding program (subprograms), program structure including subprograms is presented in the course as soon as possible after the necessary introductory material. In essence, instead of building up program structure slowly through a detailed presentation of the elementary concepts until subprograms are reached, program structure is built up as rapidly as possible to include the concept of subprograms.

This means that only the simplest form of the essential concepts, such as variable, constant, assignment, expression and process control, will be introduced before the concept of subprogram is introduced. Once the simplest form of a subprogram is introduced, all other concepts including the various data types, data structures, data manipulation techniques and more complex subprogram structures can be introduced in relation to program structure and the desired program characteristics. This kind of approach can be categorized as a "top-down" approach, since it starts with the more general concept of program structure with subprograms and uses it as the basis for the introduction of new concepts, as opposed to the "bottom-up" approach typically used in which the more basic elements are used to build up a program structure which will eventually include subprograms.

Another aspect of this approach is that the myriad of details which can be taught about any language are of less importance than the conceptualization of the overall task of programming. This is not to say that details of the language being taught are unnecessary, since the student must actually use the language in order to appreciate the task of programming; however, to take the time to teach specialized language details at the expense of teaching about the total programming task is undesirable. Once the student has the basic programming concepts, he should be able to learn the more specialized features of the language with little trouble.

The course is divided into n sections. The first four sections are concerned with the presentation of introductory material and the initial presentation of the method for designing programs. Each section

thereafter is concerned with the introduction of a set of new concepts which culminate in a programming assignment.

A diagram of the course organization is given in Figure 1. It summarizes the course by giving for each section

1) the topic
2) the steps in the method for designing programs that are supported by the section
3) the type of exercises and assignments for the section
4) the development of concepts in support of program structure
5) the specific language concepts introduced in support of subprograms

The diagram also shows where each of the five characteristics of a well-designed program can be introduced.

The philosophy of this approach is that since problem solving is a major activity in the course, motivation should be given from a problem solving point of view, and, in fact, conscious application of problem solving techniques, independent of computers, is encouraged initially as an orientation to that point of view. The course organization is intended to support this philosophy as well as focus attention on the method for designing programs.

Section I — Introductory Material : Problem Solving in General

The history of computers as well as general information about their applications is one way to begin the course and to build student interest. This information may be, optionally, spread out over the course. Whether or not this material is used in the introduction of the course,

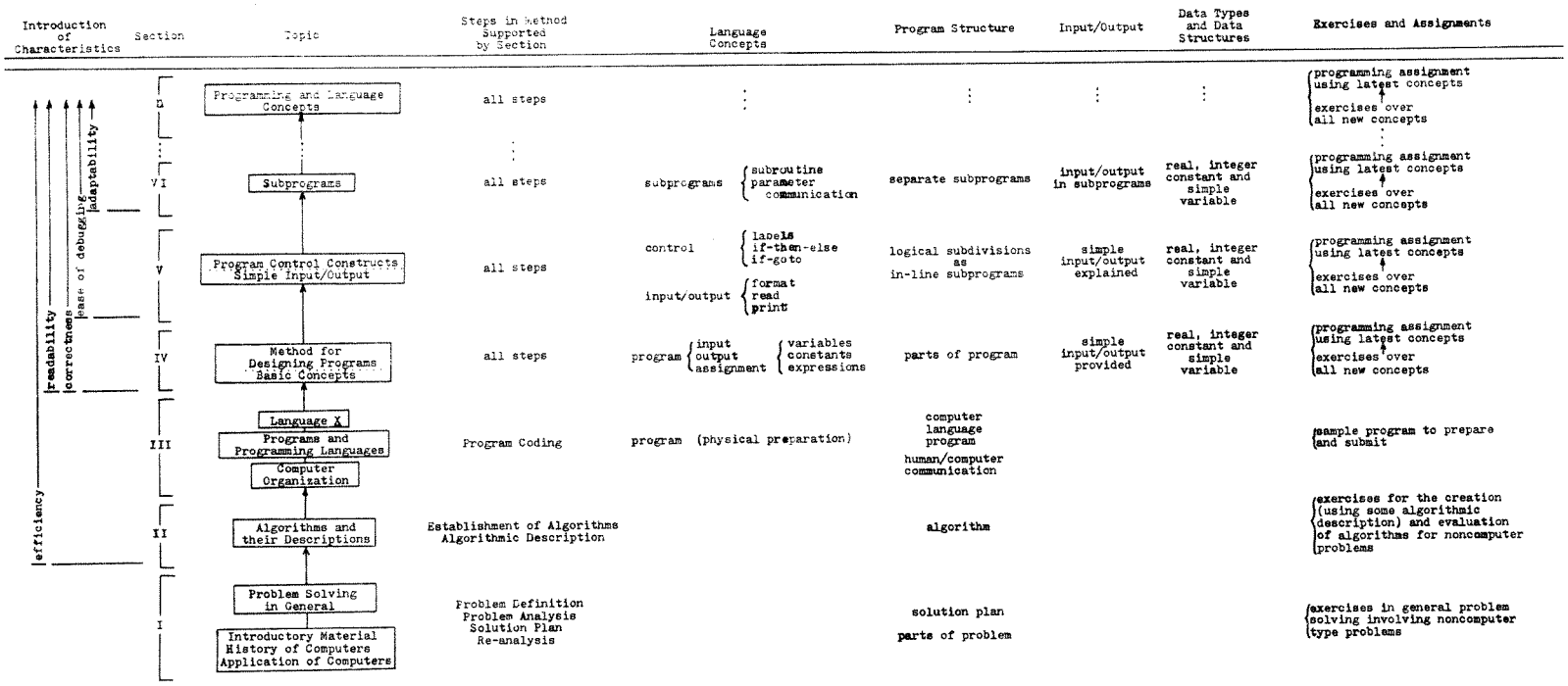
Introduction of Characteristics
Section
Topic
Steps in Method Supported by Section
Language Concepts
Program Structure
Input/Output
Data Types and Data Structures
Exercises and Assignments
efficiency
readability
correctness
ease of debugging
adaptability
Programming and Language Concepts
all steps
programming assignment
using latest concepts
exercises over
all new concepts
VI
Subprograms
all steps
subprograms
subroutine
parameter communication
separate subprograms
input/output in subprograms
real, integer constant and simple variable
V
Program Control Constructs Simple Input/Output
all steps
control
labels
if-then-else
if-goto
input/output
format
read
print
logical subdivisions as in-line subprograms
simple input/output explained
IV
Method for Designing Programs Basic Concepts
all steps
program
input
output
assignment
variables
constants
expressions
parts of program
simple input/output provided
Language X
III
Programs and Programming Languages
Computer Organization
Program Coding
program (physical preparation)
computer language program
human/computer communication
sample program to prepare and submit
II
Algorithms and their Descriptions
Establishment of Algorithms
Algorithmic Description
algorithm
exercises for the creation (using some algorithmic description) and evaluation of algorithms for noncomputer problems
I
Problem Solving in General
Introductory Material History of Computers Application of Computers
Problem Definition
Problem Analysis
Solution Plan
Re-analysis
solution plan
parts of problem
exercises in general problem solving involving noncomputer type problems

FIGURE 1

a section on problem solving in general should be included here. The emphasis at this point is on

1) Problem Definition — what things to look for in understanding a given problem
2) Problem Analysis — how to break a problem apart for analysis
3) Problem Solution — how to apply previously acquired knowledge to augment problem solution in order to get a solution plan and an answer
4) Re-analysis — how to evaluate the problem solution itself and the way in which it was found

Included here may be broad rules, such as those given by Hyman and Anderson [1965], which can be applied to various problem solving situations.

The purpose of this section is to orient student thinking toward the solving of problems without any reference to computers and introduce the concept of efficiency. In order to promote the conscious application of general problem solving methods, suitable exercises should be given to the student. These exercises need to include problems which require the student to

1) identify, in the problem statement,
   a. the given
   b. the unknown, i.e., what is the answer to the problem to be and what is its form
   c. the conditions linking the given and the unknown
2) develop a breakdown of the problem into its subproblems
3) develop a solution plan for the subproblems and thus for the problem itself; the solution to the problem may be a

specific answer or the solution plan itself, the latter of which should be emphasized in anticipation of the solution plans used in programming

4) evaluate the solution and the way in which it was found in an effort to find a "better" (more efficient, less costly) solution

5) find alternative problem representations, problem breakdowns, and problem solutions

Any problem given should be solvable without special background knowledge so that the student is not burdened by extra material. The problem, in itself, is unimportant so long as the attempt at its solution can give practice in the conscious application of techniques for solving problems. The average student should be able to solve most, if not all, of these problems so that he does not become discouraged; however, even if a student is unable to solve a problem after a concerted effort, that effort will still be of benefit to him in giving him problem solving practice and should not be seen as a failure on his part.

Suggested problem types along with points to emphasize are given below. Each type is illustrated by specific examples.

Suggested Problems for Section I

1) Simple algebra word problems expressable in single variable equations:

These can be used to give practice in the identification of the given, the unknown, and the conditions linking the given and the unknown. The analysis of some of these problems may involve diagrams to clarify

parts of the problem before the conditions linking the given and unknown can be expressed in equation form. The subproblem breakdown in this type of problem is straightforward and can be easily used to develop a solution plan which, when carried out, will produce the answer to the problem. In this case, the solution plan is an aid to finding an answer to the problem. Alternate ways of finding an answer can be explored. In problems of this type this usually means that the relationship of the variables involved can be expressed in terms of one of the other variables. There can be an advantage if the variable being solved for is the unknown in the problem.

By giving a small set of this type of problem it can be pointed out that there are certain similarities in the solution plan for each problem. These similarities can be expanded into a general purpose procedure that can be applied to any problem of this type.

Example Ia: The sum of three consecutive numbers is 99. Find the numbers.

Identify:

Given: sum is 99; sum is formed by three consecutive numbers

Unknown: three numbers - say x, y and z

Conditions: x+y+z=99, where x, y and z are consecutive

Subproblems:

(1) express equation in one variable

(2) find relation among variables that can be used for (1)

Solution plan:

(1) express two of the three numbers in terms of the third

(2) substitute that relation into the equation that links the known and unknown

(3) solve for the unknown value of the one variable

(4) use value from (3) to find the other two numbers

Solution:

(1) $x+1=y$ and $y+1=z$ therefore $y=x+1$ and $z=x+2$

(2) $x+y+z=99$ therefore $x+(x+1)+(x+2)=99$

(3) $3x=96$ therefore $x=32$

(4) $x=32$ therefore $y=x+1=33$ and $z=x+2=34$

Re-analysis:

(1) Testing the result: Since $32+33+34=99$ and since x, y and z are consecutive, the problem is solved.

(2) Alternate solution analysis: There are two other ways of finding an answer to this problem. One involves expressing the relationship of x, y and z in terms of y instead of x and the other involves expressing that relationship in terms of z. In this problem, all three work equally well.

Comment: This problem can be used as the basis of an assignment in which the student is to develop a general purpose procedure for this type of problem.

Example Ib: Develop a solution plan for the following problem statement:
The sum of three consecutive numbers is n.
Find the numbers.
Are there any conditions for which your solution plan will not work? If so, what are they?

This is good practice for the generalization of solution plans needed in programming.

2) Organizational problems related to the everyday environment:

These can be useful in showing that solving a problem does not necessarily require that there must be some mathematical equation involved.

Introductory students often try to find a single equation for the basis of their programs. Since many programs are based on a process rather than on single equation, the introduction of an acceptable answer being based on something other than an equation is desirable. The "answer" to a problem may be some form of a solution plan; this type of problem gives practice in devising such plans as the desired answer form.

While this type of problem can give practice on the identification of the given, etc., it can also be used to explore alternative ways of solving problems and used to introduce the idea of efficiency.

Example Ib: The Smith family has four sons - ages 5, 6, 7 and 8. As the older boys outgrow their jeans, the younger boys get them. On wash day, all the jeans are washed together. Sorting becomes a problem since without being measured the jeans are hard to tell apart. Devise a scheme for marking the jeans on the inside of the waistband so that they can be easily sorted. Set up your marking scheme so that when a pair of jeans is handed down, it can be remarked with the minimum amount of trouble.

Comment: This problem can be used to point out that a mathematical equation as such is not necessary for solving certain kinds of problems. The idea of efficiency can also be demonstrated by comparing the different solutions to the problem. For example, writing each boy's name on his jeans will work; however, it is not as efficient a solution as marking the oldest boy's jeans with a single mark, the next oldest's with two marks, and so forth. Then, when a pair of jeans is outgrown, it can be remarked simply by adding another mark to the existing one(s). For example,

	Oldest	Second Oldest	Third Oldest	Youngest
Solution_1	/	//	///	////
Solution_2	/			

Example Ic: There are four people in the Jones family. Each week a different person is responsible for sorting, folding and distributing the clean laundry. This week is Bill's turn and he must finish the clothes before he can go to the football game. He knows that sock sorting takes him the longest since all the clean socks are put unsorted into a big basket. Devise a sorting scheme to help Bill so he can make the kickoff. There are 6 colors, 4 sizes (one for each family member) and 2 knit patterns which must be paired.

Comment: This problem can be used to demonstrate that a solution plan rather than a specific answer can be a solution to a problem. It can also be used to expand the idea of efficiency. For example, different solution plans can be compared on the basis of how many times the socks will be looked at before sorting is complete. For example,

$Solution_1$: Pick a sock. Look for its pair by matching color, size and pattern so that each sock is looked at only once during each pass through the basket.

$Solution_2$: Sort into piles according to color. Sort each color pile into piles according to pattern. Finally sort each color-pattern pile into piles according to size.

Section II — Extension of Section I to Include Algorithms and Their Descriptions

In Section I, the concept of a solution plan was introduced. From this concept of an informal plan comes the natural extension to the formalized process — the algorithm. The purpose of Section II is to establish the need for algorithms, to define the set of rules necessary for a process to meet in order to be an algorithm, and to introduce a representation, such as the flowchart, that can be used to describe an algorithm in order that the student can create and evaluate algorithms himself.

Suitable exercises should be given which require the student to

1) use some form of algorithmic description
2) express a solution process as an algorithm; this solution process may be one given to him or one which he has created himself
3) determine whether a process is an algorithm by deciding if the process meets the necessary rules
4) follow an algorithm through its actions

As with the problems in Section I, the problems in this section may be independent of computers and should be of a familiar content. Also, they should require a nontrivial process for their solution, although they may be easily solved. Having the student (1) develop a set of rules for a problem, (2) express those rules as an algorithm and (3) evaluate a fellow student's algorithm for the same problem can be a useful exercise in both algorithm creation and algorithm evaluation.

Suggested Problems for Section II

1) Problems which require the creation of a set of instructions for "how to" do something in the physical environment:

This type of problem may require a simple physical prop that can be used by the students so that they can learn for themselves how to put the prop together and then develop a set of instructions from their own experiences. The prop may be a model car, plane, etc.

Example IIa: Devise a set of instructions for putting together a model car. The instructions should make the process of putting the car together as clear as possible. Remember, someone is going to use them to assemble the car.

Comment: This type of problem serves to point out two things. First, it points out the importance of being able to develop a set of precise rules that will be used by someone else and, especially, that the rules should be unambiguous and clearly expressed in an appropriate form such as a verbal outline or even a flow diagram. Secondly, it points out the importance of the rules being adequate to describe the entire task so that when they are followed by someone, he can do the task completely by the rules alone.

2) Problems which require the creation of an algorithm that describes some mental task:

Example IIb: Your roommate is never able to balance his checkbook so you have been doing it for him. This time you have decided that instead of doing it for him, you will provide him a set of instructions (and loan him your calculator). What instructions do you give him, assuming he can use the calculator by himself?

Example IIc: You have been hired by the Schnapps Company as an apprentice accountant. Your first job is to process the payroll. Although this is the computer age, the head of the company, Ms. Schnapps, does not believe in computers. Anyway there are only ten employees. Before she will allow you to handle the payroll and write the checks, you must first write up how you are going to process the payroll and let her approve your plan. Remember, Ms. Schnapps likes efficient people working for her. Things you need to know:

1) There are 10 employees (including Ms. Schnapps) who each work 40 hours per week
2) Each person receives a weekly check based on length of employment according to the following pay scale:
   a) Less than 6 months: $1.50/hour
   b) More than 6 months but less than one year: $2.00/hour
   c) For each year over 1 year but less than 10 years: add $.25/hour

d) For more than 10 years but less than 50 years: weekly salary is $200/week
e) For 50 years or more: salary is $500/week (guess who?)

3) Each employee has the following deducted from his check:
a) income tax: 10% of gross salary
b) social security: 5% of gross salary up to $10,500/year; gross salary in excess of $10,500/year is not subject to social security deductions.

Comment: Both IIb and IIc require the student to produce a more formalized process than does IIa. Simple iteration and simple decisions can be introduced at this time. Description of the processes may be a precise verbal outline or flowchart; the latter is probably better in anticipation of its use later in the course. The student may be required to trace through at least two algorithms that are supposed to accomplish the same task in order to establish

1) if each is an algorithm
2) if each accomplishes the task
3) which one is the more efficient based on the number of steps involved, etc.

Section III — The Introduction of the Computer; the Relationship of Algorithms, Programs, and Programming Languages

The purpose of Section III is to introduce

1) the basic concepts of computer organization, such as the parts of the computer and how they interact
2) the ways humans and computers interact through
a. programs
b. languages
c. devices

3) the relationship between algorithms, programs, and programming languages
4) the practical aspects of computing such as
   a. the physical representation of a program
   b. the submission procedures
5) the particular programming language being used, to the extent that the student can run a sample program

These are introduced in order to establish the necessary groundwork for the introduction of the method for designing programs. The only exercise required of the student is to prepare a sample program and to run it so that he will become familiar with the physical tasks involved. The program itself should be a solution to a simple problem so that both the problem and the program can be used in Section IV to illustrate the method for designing programs.

Suggested Programming Assignment for Section III

The sample program used for Sections III and IV should require only the language concepts of program, assignment statement, simple variable (real and integer), constant (real and integer) and simple expressions. Input and output should be of the simplest forms. These restrictions limit the sample program to being strictly linear, i.e., no jumps. One type of problem that can be used at this point is one which requires the application of a straightforward mathematical formula on a limited amount of data.

Example IIIa: Write a program to read in the weights of five people to the nearest hundredth of a pound. Find the average weight to the nearest tenth of a pound. Output the five weights and their average.

Section IV — Presentation of the Method for Designing Programs : Basic Concepts

The purpose of this section is to orient student thinking toward problem solving using computers by expanding the four steps from Section I to accommodate the special tasks which characterize the designing of computer programs. The method for designing programs is presented in its entirety, that is all ten steps are described, in as much detail as can be given in light of the material presented to the student up to this point. The details omitted during this presentation will be presented in later sections of the course as supporting concepts are introduced which allow natural expansion of the method. For example, subprograms and their expansive effect on the method will normally be encountered after more basic concepts such as variable, constant and assignment have been learned by the student.

The sample program from Section III may be used at this time to illustrate the ten steps in the method. The statement of the problem, for which the program is a solution, should be given as the starting point so that each step in the method can be individually explained in sequence in order that the student can see the natural progression from Problem Definition through Re-analysis. Special care should be taken to show how the parts of the given problem (i.e., the input, the output, and the relationship between the input and output) are reflected in the program itself. This is important in order to establish the idea that different parts of the program are responsible for certain logical functions which are directly related to the problem and its definition.

The sample program can also be used to introduce the characteristics of correctness and readability. Correctness can be motivated by showing how the function described by the program will give the desired outputs for legal inputs. Readability can be motiviated by the use of variable names which describe themselves and by the use of spacing and commentary in the program body.

Specific language concepts needed for this section include the following:

1) program
2) assignment statements
3) simple variables (integer and real)
4) constants (integer and real)
5) expressions
6) simple input and output, where these are given to the student to use without detailed explanation.

It should be emphasized to the student that this method for designing programs will

1) be expanded as the course progresses
2) provide the framework for the introduction of the rest of the course content
3) be the pattern he will be using in his own programming assignments

This section is the first to include exercises over new concepts and a programming assignment for the student to write himself so that he can practice the method for designing programs in its present stage and apply the new concepts of this section. Suggested exercises and programming assignments are given below.

Suggested Exercises for Section IV

1) Problems which require identification and interpretation of the new concepts:

E1a) Problems to identify the correct syntactic forms of variables and constants (real and integer)

E1b) Problems to evaluate expressions (real and integer)

E1c) Problems to evaluate assignment statements, singly and in sets

2) Problems which require creation of the new concepts:

E2a) Problems to create expressions, given algebraic equations

E2b) Problems to create a series of assignment statements, given a verbal statement or flow diagram of some process

Suggested Programming Assignment for Section IV

Due to the limited number of language concepts available to the student at this time, the programming assignment is limited to those programs which are linear, i.e., do not contain control statements. Input and output should be provided for the problem.

One type of problem that requires only the available language concepts is a conversion problem in which a value is converted from one system to another. Two examples are given below.

IVa: Time Conversion [Maurer and Williams, 1972, p. 7]

Write a program that will read a five digit positive integer representing an amount of time in seconds. Convert this to hours, minutes and seconds. Output should be of the form

___ seconds = ___ hours ___ minutes ___ seconds

where the program fills in the blanks in the form. Use the following statements for input and output:

(input statement)
(output statement)
(data card)

IVb: Time Difference [McCracken, 1972, p. 50]

Write a program that will accept two integer numbers, each of which represents the time in hours on a 24 hour system. Express the difference between the two numbers in hours and minutes. Assume that the first number is always earlier than the second number. Output should be of the form

The difference between $\underline{time_1}$ and $\underline{time_2}$ is

___ hours and ___ minutes

where the program fills in $time_1$, $time_2$, and the blanks. Use the following for input and output:

(input statement)
(output statement)
(data card)

Example:

The difference between 1130 and 2200 is

10 hours and 30 minutes

It is expected that the assignment will require the student to use the method for designing programs; therefore, there should be some evidence other than the completed program which indicates his work. Simple documentation as well as the program being efficient, readable and correct should be required.

Section V — Program Control Constructs and Simple Input/Output

This section is used to build up the student's knowledge of basic language concepts and to expand the idea that parts of a program reflect the logical subdivisions of a given problem. The characteristic, ease of debugging, is introduced in this section.

Basic language concepts needed at this time center around the programming concept of process control. The simpler forms of control statements, such as if-then-else or if-goto, are necessary. This type

of statement allows for a more complex program structure which can be used to demonstrate how the conditions and subproblems in a given problem can require that decisions be made within the program itself in order to regulate the execution of the logical divisions, (or in-line subprograms), of the program. Rather than having a strictly linear execution of the parts of the program, execution is typically nonlinear, and this type of program structure is to be reflected in the physical representation of the program.

The other language concepts explained in this section are simple input and output statements such as those used in Section III and Section IV. Details for input and output should be only enough for the student to input his own data and output the program results with simple headings. The data types for input and output should be restricted to those with which the student is already familiar.

Ease of debugging can be motivated in this section through the use of output of both test values (such as echo print of input) and error messages.

The types of errors that are normally encountered when using the language concepts of this section are to be explained, and all new concepts are to be related to the characteristics of efficiency, readability, correctness and ease of debugging.

Suggested Exercises for Section V

1) Problems which require interpretation of the new concepts:

E1a) Problems to determine the flow of control through a series of statements which include control statements; these can ask the student to determine values for certain variables at the end of a series of statements or ask the student to create a flowchart that reflects the code of the statements

E1b) Problems requiring the student to evaluate the efficiency of a set of statements which include control statements; e.g., determine number of statements executed, etc.

E1c) Problems requiring the student to interpret input and output statements; e.g., given a set of input statements and data cards, indicate the values for each variable in the input list; given a set of output statements and values for the variables in the output list, show how the output will look

2) Problems which require the creation of the new concepts:

E2a) Problems requiring the student to create a series of statements including control statements that will reflect a given flow diagram

E2b) Problems requiring the student to create a series of statements including control statements that will be the solution to a given problem

E2c) Problems requiring the student to create input and output statements to meet specific conditions; e.g., given that the data cards are of a certain form, create the proper input statement

E2d) Problems requiring the student to create the input and output for a given problem; e.g., the student makes his own decisions as to the layout of data cards and output listing and creates the necessary input and output statements; the student determines a series of output statements that are useful in testing and debugging a program

Suggested Programming Assignment for Section V

The programming assignment for this section should require two things:

a) the program should require generalization of input, i.e., it should process multiple sets of data or it should provide for the input of initial values

b) the program should require the use of conditionals so decision making and iteration in the program is done other than for looping to input more data.

1) Problems involving the application of a mathematical formula:

Va: Quadratic Equation [Maurer and Williams, 1972, p. 18]

> Given a set of three numbers a, b and c, determine the solutions to the equation $ax^2+bx+c=0$ as follows:
> If b^2-4ac is less than 0, then the equation has no solution and an appropriate message should be given; if $b^2-4ac=0$, the only solution is $-b/2a$; if b^2-4ac is greater than 0 then there are two solutions, $(-b+\sqrt{b^2-4ac})/2a$ and $(-b-\sqrt{b^2-4ac})/2a$.
> The program should process n sets of a, b and c.

Comment: The introduction of library routines can be motivated by this problem, explicitly, the square root function. Also, this problem statement, like most others, is ambiguous; for example, what if a = 0? The student should find such ambiguities during problem analysis and develop a satisfactory resolution for them; for example, testing for a = 0 in his program.

Vb: Triangles [Maurer and Williams, 1972, p. 17]

> Write a program that will read three numbers (representing the lengths of the sides of a triangle) and print out one of the following four words:
>
> | | |
> |---|---|
> | NONE | if the three lengths do not represent the sides of a triangle (i.e., if the sum of the lengths of any two sides is not greater than the third side) |
> | GENERAL | if the lengths specify a general triangle (i.e., one whose sides are all of different lengths) |
> | ISOSCELES | if any two sides are of equal length |
> | EQUILATERAL | if all three sides of the triangle are of equal length |
>
> Write a program so that it will print out the lengths of the three sides along with the type of triangle they represent. The program should process an arbitrary number of sets of three numbers and will terminate upon reading a set of three numbers all of which are zero.

Comment: The type of data used for this program should be carefully considered. Integer values for the lengths of the sides present no problem. However, if the values represent physical measurements and contain fractional parts, i.e., are decimal values, then some decisions must be made as to when two values are considered equal. For example, is 4.325 to be considered equal to 4.33?

2) Problems involving the output of tabular information:

Vc: Mortgage Table [Maurer and Williams, 1972, p. 9]

Given the amount of a mortgage, the rate of interest being charged, the monthly payment, and the amount of tax due on the property (per year), produce a table of the principal, interest, tax, payment, and the principal left for each month of the mortgage's existence. For example, if the principal is $20,000.00 the interest is 6%, the tax per year is $700.00, and the payment is $300.00 per month, then the output should be

Month	Principal($)	Interest($)	Tax($)	Payment($)	New Principal($)
1	20,000.00	100.00	58.33	300.00	19,858.33
2	19,858.33	99.29	58.33	300.00	19,715.95
.					
.					
.					

Note: The last payment on the mortgage may be less than $300.00.

Comment: Calculations for this type of problem require that the interest rate be expressed in terms of the period of payment. Therefore, the interest rate used should be on a per-month basis, i.e., interest per month = interest per year/12.

Vd: Conversion Tables [Maurer and Williams, 1972, p. 21]

Prepare a table with two columns, the first representing miles and the second kilometers, using the relation 1 mile = 1.61 kilometers. The table should contain for

1,2,3,...,100 kilometers the corresponding number of miles, and for 1,2,3,4,5,...,65 miles the corresponding number of kilometers. Both the first and second columns must contain numbers in ascending order.

Notes and Explanations: The table should start as follows:

Miles	Kilometers
0.62	1.00
1.00	1.61
1.24	2.00
1.86	3.00
2.00	3.22
.	.
.	.
.	.

Set up the program so that the range for miles and kilometers is flexible. Provide for these ranges to be input to the program.

Comment: The accuracy of the table depends upon the accuracy of the conversion relation. The accuracy of this table can be improved by using 1 mile = 1.609344 kilometers.

Section VI — Subprograms

This section introduces the concept of a stand-alone subprogram. The characteristic of adaptability is introduced with the emphasis on program modularity. With the inclusion of subprograms, program structure has been built up to the point where the major force of the method for designing programs, i.e., the creation of subprograms for a program based on the breakdown of the given problem into subproblems and their relation to the characteristics of a well-designed program, can become effective in developing well-designed programs.

The language concepts needed for this section include the simplest form of subprogram found in the specific language being used. It is desirable to minimize the complexities of parameter communication as much

as possible. This can be done by giving the student a set form to use with parameters (if they are used), such as the parameters in the call and in the subprogram list must be the same. Or, he can be given a form without parameters, such as a common block form where the COMMON statement must appear exactly the same in both the main and subprograms. This allows emphasis to be placed on program structure rather than on specific details which are elaborated on in later sections.

Adaptability of a program is the last characteristic to be introduced. From this section on all new concepts should be related to all five of the characteristics for a well-designed program.

Suggested Exercises for Section VI

1) Problems which require the interpretation of the new concepts:

E1a) Problems which require the student to identify legal subprogram communication; e.g., given a call and a subprogram, determine if the communication is set up correctly

E1b) Problems which require the student to trace through a program containing subprograms; e.g., given a program and its subprograms, require the student to determine the values sent to and returned from the subprogram

E1c) Problems which require the student to determine if a program with subprograms is a solution to a given problem; e.g., given a problem statement and a description of a program and its subprograms which outlines their purpose and their inputs and outputs, determine whether the program contains all of the parts needed to be a solution to the given problem and also determine if there is a better program/subprogram structure for the problem.

2) Problems which require the creation of the new concepts:

E2a) Problems which require the student to create the syntax needed for communication between calling program and its subprograms; e.g., given a set of parameters, develop the necessary program statements for both main and subprograms

E2b) Problems which require the student to write a main program and its subprograms given a verbal description or a flow diagram

E2c) Problems which require the student to write a main program and its subprograms for a given problem

Suggested Programming Assignment for Section VI

Since this assignment is intended to focus on the use of subprograms, the problem should contain several obvious subproblems. The problem should not require any language concepts other than those already introduced. One type of problem that can be used here is that which requires the computation of several formulae for tabular output.

VIa: Monthly Payments [Federighi and Reilly, 1971, p. 21]

The exact formula for calculating the monthly payment R required to pay off a loan of P dollars in N years when an interest rate i is charged is:

$$R = \frac{P}{12}\left[\frac{i\left(1+\frac{i}{12}\right)^{12N}}{\left(1+\frac{i}{12}\right)^{12N}-1}\right]$$

Two approximations which a prospective borrower could compute on a slide rule are:

$$R_{a1} = \frac{P}{12}\left[\frac{i}{1-\exp(-iN)}\right] \quad \text{and} \quad R_{a2} = \frac{P}{12}\left[\frac{i}{1-\exp(-iNx)}\right]$$

$$\text{where } x = 1 - \frac{i}{24}$$

The borrower, however, wants to know how accurate these approximate formulas are.

Write subprograms to compute each of these as functions of P, i, N. Write a main program to read in P, N, i_1, Δi, i_2 and printout a table of each of the three R's vs. i from i_1 to i_2 in steps of Δi. Print the input values of P and N in the heading of the table (see sample table below).

MONTHLY PAYMENT FOR XXX.XX DOLLARS AND XX YEARS

INTEREST RATE	EXACT PAYMENT	1ST APPROXIMATION	2ND APPROXIMATION
.01	XX.XX	YY.YY	ZZ.ZZ
.02	.	.	.
.03	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Comment: It may be desirable to change the wording of the problem so that the student is to decide what parts of the problem should be subprograms rather than stating it explicitly for him.

Sections VII - n — Expansion of the Method for Designing Programs Through the Introduction of New Programming and Language Concepts

At the end of Section VI, the majority of the groundwork needed for further expansion of the method for designing programs has been laid. This groundwork includes

1) program structure with subprograms
2) the five characteristics for a well-designed program
3) the method for designing programs to the point where subproblems can be related to subprograms

The remaining sections of the course are concerned with the expansion of the method for designing programs through the introduction of new programming and language concepts. These concepts include various data types and data structures, detailed aspects of subprograms and program structure, and various techniques for data manipulation including the complexities of input and output.

The pattern of each section is as follows:

1) a set of concepts is presented within the framework of the method, i.e., each concept is explained in relation to the method and in relation to its affect on the desired program characteristics
2) a set of exercises over the concepts is given to the student for practice before he actually tries to write a program using them
3) the set of concepts presented is the basis for a programming assignment given to the student so that he may practice the method for designing programs (this assignment is given after he has performed satisfactorily on the exercises.)

The purpose of these sections, then, is to fill the details into the framework of the method for designing programs and to allow the student to use the method in conjunction with the latest concepts. In this way the student is required to use the concepts he has learned as well as to use the method for designing programs in its latest expanded form.

3.4 Presenting New Concepts to the Student

In order to utilize the framework of the course structure to full advantage, it is necessary that new concepts be presented to the student within that framework. First, a new concept is defined and related to previously introduced concepts that are prerequisite to it. Once the student has mastered the basics of the concept, the relationship of the concept to the method for designing programs can then be discussed This discussion does two things. One, it relates the new concept to each

step in the method to which it is applicable. Two, it describes the effect of the new concept on the characteristics of a well-designed program. This discussion is to provide relevant information for the student to utilize during the subsequent programming assignment, which is given at the end of the section containing the new concept, and for future use as well.

Concept presentation can be approached from the point of view that each new concept is a new tool for the student to add to his programming repertoire and that it will be useful to him in one of two ways:

1) it allows him to do something he was unable to do previously
2) it allows him to do something in an alternative although not necessarily equivalent or equally applicable, way from the way he can already do it.

The introduction of the concept of subprogram provides an example of the first way. Before the student knew about subprograms, he could only produce a program whose structure was an undivided whole into which all parts of a problem fit. With the introduction of subprograms the student can construct a program with a modular structure reflecting the subproblems of the given problem more directly.

An example of the second way comes from the presentation of specific types of subprograms. Suppose that the language being used allows for two kinds of subprograms — subroutines and functions. Either can be used in a program as a subprogram; however, the choice between them is based on decisions concerning the required logical function of the subprogram and its communication with the rest of the program. The new concepts of subroutine and function can be presented to the student, then, as

being two different ways of creating a subprogram, each one having its own particular characteristics and effects in a program.

3.5 Requiring the Student to Demonstrate his Knowledge of Concepts

As previously stated, before the student is given a programming assignment in which several new concepts are to be used, he should demonstrate how well he understands the individual concepts. This demonstration is in the form of exercises which should require two things:

1) that the student be able to interpret the mechanics of the concepts, i.e., when given examples he can identify the concept, he can determine whether the example is correct as well as tell what is being done in the example; for example, evaluation of expressions, output of arrays, the division of a problem into logical subproblems, etc.
2) that the student be able to use the concept, e.g., create language constructs representing the concept

The rationale for requiring the student to work these exercises is that a program is in itself a complex problem for the student, especially if it is in the programming assignment where he is first required to use new concepts. This is, to a degree, learning by trial-and-error about the new concepts. It is advisable that the student understands the concepts before he tries to combine them; otherwise, any misconceptions on his part may cause him to waste a great deal of time and effort, to ignore the concept if he can program around it, or to use it incorrectly and not know it. Only when the student has demonstrated that he understands

and can use the concepts is he given a programming assignment which requires him to combine the latest concepts (plus those previously learned) to produce a program.

3.6 Requiring the Student to Demonstrate the Method for Designing Programs

There are two kinds of programming assignments that are useful for the practice of programming skills. The first type is the traditional one in which the student is required to write and run a program. The second type is one in which the student is required to produce a detailed solution plan in lieu of writing the complete program. The applicability of the two types of assignments within the course and the kinds of programming problems for each type are discussed below.

Construction of a complete program

The traditional type of programming assignment is applicable in any section of the course. In order to gain practical experience using programming and language concepts and to practice the method for designing programs, the student must write and run programs. The programming problems for this type of assignment should meet the following criteria:

1) the problem should make use of all new concepts presented so that the student will have practical experience working with them either as
   a. programming or language concepts to be used in writing the program (e.g., control statements, parameter communication, etc.), or
   b. the subject (application) of the problem (e.g., Polish notation, expression evaluation for an interpreter, etc.)

2) the problem should have as its subject (application) one of the following:

   a. some topic which is familiar to the student (such as payrolls, games, lexical analysis) or is of interest to him (such as problems from his major field). Unless students are known to possess adequate mathematical background, problems of numerical applications, other than those which are straightforward, should be avoided. Any other subject which cannot be easily taught should be avoided as well; otherwise, valuable time must be spent supporting an unfamiliar topic sufficiently for it to be programmable by the student. Many game-oriented problems are complex enough to provide good programming practice while being familiar to the student [Ralston, 1971, p. 459].

   b. some topic which is computer science oriented, e.g., Polish notation, expression evaluation for interpretation, etc.; this kind of topic can be used after the student has become familiar with basic concepts

3) the problem should require nontrivial application of the method for designing programs so that the student works through all the steps in the method

This type of programming assignment should require evidence that the student has worked through all of the steps in the method. It would be desirable to monitor the student's progress by having him submit the following for approval for each assignment:

1) an outline of his solution plan(s)

2) the algorithmic development of his solution plan

3) the coded program along with the tentative test data and procedures to be used in running the program

4) the tested and debugged program with

   a) internal and external documentation

   b) record of errors made giving type of error, reason for error, and the structure associated with it

   c) re-analysis of program, errors and new concepts

Unfortunately, the typical course environment does not allow for constant monitoring of the student's progress so some compromises must be made. The first three items listed above can be omitted and compensated for by assigning them in the shorter exercises so that the student will get practice in these three important parts of the programming task. An exercise covering one of the following can be assigned in the various sections of the course for problems less complex than those in the programming assignment:

a) the creation of a solution plan

b) algorithmic development

c) construction of test data and procedures

The tested and debugged program must, however, be required along with some kind of documentation. If the re-analysis step is to be useful, either the student must hand in something for instructor commentary or re-analysis must be discussed in class after each programming assignment. The former is preferable; otherwise, the student will not be motivated to do it. The record of errors can be useful in focusing the student's attention on his

programming habits and can play an important part in the re-analysis step.

There is a limitation on the size and complexity of the programming problem that can be assigned to the introductory student due to his lack of knowledge of programming and language concepts and to the short length of time that can be allowed for programming assignments. The programs, therefore, tend to be rather simple ones and do not reflect the problems that are encountered during the construction of large programs. These simple programs fail to motivate the student to use all the tools, such as flowcharting and debugging aids, that he has at hand since he can often hold much of the program in his head. It is not unusual for a student to complete a program, even a relatively lengthy one, and then flowchart it. If the student is to be prepared for the kinds of programming tasks that he will encounter either in a future course or in a job, he should be able to apply the method for designing programs to large programs as well as to small ones.

Construction of the top level design for a program

In order to bridge the gap between what the student can do with the limited tools he has and what he should be prepared to do, the programming assignment that requires him to produce a detailed solution plan for a program is proposed. In essence, the student is required to design the top levels of a large program. The assignment should require that

1) inputs and outputs be adequately defined for the program
2) program and subprogram structure be such that all parts of the problem are included

3) inputs and outputs for subprograms be adequately defined
4) program structure contribute favorably to the desired program characteristics
5) program testing procedures be outlined with descriptions of both test procedures and test data

While the student is not working through all of the steps in the method for designing programs, he is having to use most of them in order to construct such a top level design. Especially important in this type of assignment is the necessity for the student to analyze the given problem adequately. The removal of the coding, testing and debugging elements frees the student from the constraints of his limited knowledge of specific language concepts and allows him to spend more time on the analysis of the problem and on the construction of a solution plan.

Problems suitable for this type of programming assignment should have topics similar to those for the traditional programming assignment; however, the complexity of the problem should be such that the student is unable to keep its whole organization in his head at one time.

This type of assignment has a special nature. It should be given after the student has acquired a working knowledge of subprograms and the more complex control structures, a basic understanding of the characteristics which affect the design of a program, and some practical programming experience. This type of assignment is ideal as the programming assignment for a section on program structure which contains an analysis of the affects of the method for designing programs on program structure and the relationship of the desired program characteristics to program

structure. This kind of section, encountered in the latter part of the course, can reestablish the overall picture of the programming task for the student, as he is being drawn into the details of program coding, while providing the necessary support for the task of designing a large program.

Two examples of problems suitable for this kind of an assignment are given below. The general instructions for this type of problem should require the student to define a main program and associated subprograms, giving for each one

1) the general function performed
2) the input data required
3) the outputs required
4) the subprograms called

The specific details of data representation are not required; however, the overall program structure should be given a suitable representation, e.g., a module chart.

Example 3.6a: Monthly Account Statement [Maurer and Williams, 1972, p. 79]

Each entry ("record") of the file of a department store for customers with charge accounts contains

a) An account number: a six-digit positive integer
b) Name and address of customer: a sequence of up to 50 characters.
c) Present balance: a positive number for debits (i.e., if customer owes money to the store), a negative number for credits.

Assume that this file is available on punched cards, one card per customer. Assume further that the file is in no particular order ("unsorted"). For each purchase made by a customer a "charge card" is punched containing (a) the account number, (b) the word CHARGE, and (c) the amount of purchase, a positive number. For each payment made by a

customer a "payment card" is punched containing (a) the account number, (b) the word PAYMENT, and (c) the amount paid, a positive number.

Using the charge cards, the payment cards, and the customer file as input, write a program that will (a) print a monthly statement for each customer and (b) print the updated customer file.

Note that for each customer no charge card or payment card can be present or that one or more charge cards or payment cards can be present. The monthly statement for each customer should show (a) the account number, (b) the name and address of the customer, (c) all payments, (d) all charges, (e) previous balance, and (f) new balance. Include in the charges a 1.5% interest and handling fee based on the balance in the customer file, if that balance is positive.

You may assume that the number of customers does not exceed 30 and that the number of charge cards and payment cards does not exceed 100 each.

Example 3.6b: High Card

Write a program which will play the card game of High Card. High Card is played according to the following rules:

1) A shuffled deck of cards is dealt to four players.
2) The dealer (one of the four players) begins the game by playing any card in his hand.
3) Each player in turn plays a card in the same suit if possible.
4) If a player has no cards in the proper suit then he may play any card in his hand; however, he cannot win the "trick."
5) When each player has played one card, the winner of the trick is determined as being the player who played the highest card in the proper suit, i.e., that of the first card played in the "trick."
6) The four cards played are set aside.
7) Play continues as outlined above until all 13 "tricks" (all 52 cards are played) with the winner of each trick playing first on the next one.
8) The game is won by the player who wins the most tricks.

Comment: Any ambiguities should be resolved by the student, and he should record the assumptions that he makes.

CHAPTER 4

CONCLUSION

It has been the intent of this paper to present an organizational framework for the programming task which can be taught to the introductory student and utilized by him at the elementary and advanced levels of programming. Such a framework is embodied in the proposed method for designing programs. The introduction of the method at the introductory level of a computer science curriculum has positive implications for the student; these were discussed previously. Besides affecting the student, the method for designing programs can have a positive affect on the remainder of the curriculum. A brief summary of the advantages for the student is given below. This summary is followed by a more detailed discussion of how the proposed method can be used in the more advanced courses in the curriculum.

4.1 Advantages for the Student

The student who has the method for designing programs available to him during the introductory course has several advantages. Among them are the following:

1) he does not have to create such a method for himself
2) he has an approach that he can use for future programming projects
3) he should be able to produce a better quality program
4) he has a working framework in which to assimilate the new programming techniques and computer concepts that he will encounter

5) he can add to or alter the basic method consciously through re-analysis of his own programming experiences so that the method suits his particular situation

4.2 Using the Method for Designing Programs in Advanced Courses

All courses which follow the introductory course can assume that the student has a conscious awareness of the overall programming task and that he is familiar with the characteristics that are desirable in programs. The method for designing programs can be especially useful in courses in which programming is a major part of the course or in which programming is used as a vehicle to demonstrate parts of the course content. These courses can use the method as a framework for the structuring of the course content or as the basis for an explicit method to be used for special classes of programs.

Two types of courses are given below as examples of how the method for designing programs can be used in the more advanced courses. In the first example, the method is used primarily to structure the course content. In the second example it is focused on one specific class of programs.

The first type of course is one in which a specific aspect of programming is explored in detail and various classes of programs are used as programming assignments in the exploration of that aspect; such courses as Course I1 — Data Structures, Course I2 — Programming Languages and Course B4 — Numerical Calculus as described in "Curriculum 68" [ACM, 1968], fall into this type of course.

For a specific example, the method for designing programs is applied to a data structures course which explores in depth various kinds

of data structures that are used to represent information in programming problems. By using the method for designing programs as the framework for the course content, each new data structure and the associated techniques for manipulating it can be related to the entire programming task. For instance, a stack structure, once defined, can be related to programming by describing, among other things,

1) the characteristics of a programming problem which indicate that a stack is to be used
2) the characteristics of a solution plan that indicate the use of a stack
3) the various ways that a stack can be represented algorithmically
4) the ways that a stack can be implemented in various languages and the ways that the stack structure has been implemented in the languages that have a stack feature
5) the specific characteristics of a stack which must be tested for when it is used
6) the typical errors encountered when implementing a stack and when using it in a program
7) how the stack affects the characteristics of a well-designed program

In this course the presentation of the various data structures is analogous to the presentation of new concepts as described for the introductory course. The presentation of the different data structures as being alternative ways of representing the information given in a programming problem should receive strong emphasis, especially how the alternatives can affect program design.

Due to the depth of exploration of the various data structures and their relationships with programming languages, computer storage structures and manipulation techniques, an organizational framework is important in order that the many details be perceived in proper relation to the overall programming task in which they are used. Since it is assumed that the student is already familiar with the method for designing programs from the introductory course, it can be used for the organizational framework, thus precluding the development of such a framework especially for this course. It is also assumed that the student is able to apply the method for designing programs to the programming assignments in the course.

The second type of course is one in which a specific class of programs is studied; such courses include Course I5 — Compiler Construction and Course A6 — Computer Graphics as outlined in "Curriculum 68" [ACM, 1968]. For this type of course, the method for designing programs can be focused on the special features found in the specific class of programs being studied.

For example, in a compiler course any given programming problem which requires that the solution be a compiler will have specific characteristics that must be identified and analyzed. For instance, the Problem Definition step and the Problem Analysis step can be focused on the specific aspects of source languages that should be separated out, e.g., storage allocation and expressions. The Solution Plan step can be focused on the parts of compilers which handle the various aspects of the source language, and the Program Coding step can specify ways to implement the parts of a compiler.

In this course, the refined method for designing programs serves as a guide for the construction of a specific class of programs as opposed to the basic method presented in the introductory course that serves as a guide to program construction in general. Again the framework of the method can provide an organization for the course content to the extent that the student does not lose sight of the total programming task while he is involved in the details of writing compilers.

4.3 Summary

Classroom experiences and student commentary as observed by the author as well as recent articles, such as those by Gries [1974] and Kernighan and Plauger [1974], indicate that there is a need for more emphasis on problem solving and program quality, especially at the introductory levels of the curriculum. The proposed method for designing programs is intended to increase the emphasis on these two areas of concern. While the proposals in this paper have not yet been applied to the classroom, it is anticipated that they will be of significant value there. Incorporation of the proposals into an introductory course is planned for the latter part of this year in an experimental team teaching project. The results of this project will be used as the basis for further refinement of the proposed method for designing programs and for extending its application to the more advanced courses of the curriculum.

BIBLIOGRAPHY

ACM Curriculum Committee in Computer Science (1968) Curriculum 68. CACM 11(3), 151-197.

Baker, F.T. (1972) Chief Programmer Team Management of Production Programming. IBM Systems Journal 11(1), 56-73.

Dijkstra, Edsger W. (1969) Notes on Structured Programming. Report EWD249, Tecnical University Eindhoven, The Netherlands.

_____ (1971) A Short Introduction to the Art of Programming. Report EWD316, Technical University Eindhoven, The Netherlands.

Federighi, F.D. and E.D. Reilly, Jr. (1971) Computer Science Laboratory Exercises. Schenectady, New York: Reidinger and Reidinger, Limited.

Forsythe, A.I., T.A. Keenan, E.I. Organick, and W. Stenberg (1969) Computer Science: A First Course. New York: John Wiley and Sons, Inc.

Gagné, Robert M. (1971) The Conditions of Learning, Second Edition. New York: Holt Rinehart.

Gray, Max and Keith R. London (1969) Documentation Standards. Princeton: Brandon Systems Press, Inc.

Gries, David (1974) What Should We Teach in an Introductory Programming Course? SIGCSE Bulletin 6(1), 81-89.

Hadamard, Jacques (1945) An Essay on the Psychology of Invention in the Mathematical Field. New Jersey: Princeton University Press.

Hoare, C.A.R. (1969) An Axiomatic Basis for Computer Programming. CACM 12(10), 576-583.

Hyman, R. and B. Anderson (1965) Solving Problems. International Science and Technology, September, 36-41.

Kernighan, B.W. and P.J. Plauger (1974) Programming Style. SIGCSE Bulletin 6(1), 90-96.

Knuth, Donald E. (1969) Fundamental Algorithms. Reading, Massachusetts: Addison-Wesley Publishing Company.

Maurer, H.A. and M.R. Williams (1972) A Collection of Programming Problems and Techniques. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Maynard, J. (1972) Modular Programming. Princeton: Auerbach Publishers.

McCracken, Daniel D. (1972) A Guide to Fortran IV Programming, Second Edition. New York: John Wiley and Sons, Inc.

Merrill, M. David (1971) Necessary Psychological Conditions for Defining Instructional Outcomes. Educational Technology 11(4), 34-39.

Nassi, I. and B. Shneiderman (1973) Flowchart Techniques for Structured Programming. SIGPLAN Notices, August, 12-26.

Naur, Peter (1969) Programming by Action Clusters. BIT 9, 250-258.

_____ (1972) An Experiment on Program Development, BIT 12, 347-365.

Neely, Peter M. (1973) On Program Control Structure. Proc. of the ACM 1973, 119-125.

Newell, Allen and Herbert Simon (1972) Human Problem Solving. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Nilsson, Nils J. (1971) Problem Solving Methods in Artificial Intelligence. New York: McGraw Hill Book Company.

Parnas, D.L. (1971a) Information Distribution Aspects of Design Methodology. Report, Carnegie Mellon University, Pittsburgh, Pennsylvania, February.

_____ (1971b) A Paradigm for Software Module Specification with Examples. Report, Carnegie Mellon University, Pittsburgh, Pennsylvania, March.

_____ (1972a) A Course on Software Engineering Techniques. Proc. SIGCSE Technical Symposium, March, 154-159.

_____ (1972b) Some Conclusions from an Experiment in Software Engineering Techniques. Report, Carnegie Mellon University, Pittsburgh, Pennsylvania, June.

_____ (1972c) Use of the Concept of Transparency in the Design of Hierarchically Structured Systems. Report, Carnegie Mellon University, Pittsburgh, Pennsylvania, July.

_____ (1972d) On the Criteria to be Used in Decomposing Systems into Modules. CACM 15(12), 1053-1058.

Polya, G. (1957) How to Solve It. Garden City, New York: Doubleday and Company, Inc.

Ralston, Anthony (1971) Introduction to Programming and Computer Science. New York: McGraw Hill Book Company.

Sackman, Harold (1970) Man-Computer Problem Solving. Princeton: Auerbach Publishers, Inc.

Simon, Herbert A. (1969) The Sciences of the Artificial. Cambridge, Massachusetts: The M.I.T. Press.

Weinberg, Gerald M. (1971) The Psychology of Computer Programming. New York: Van Nostrand Reinhold Company.

Wirth, Niklaus (1971) Program Development by Stepwise Refinement. CACM 14(4), 221-227.

_____ (1973) Systematic Programming: An Introduction. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.