

A Programming Language For the Teaching  
of Algorithmic Analysis

by

William Max Ivey and Larry C. Eversole  
University of Houston, Houston, Texas

## Introduction

This report deals with the design, implementation and evolution of programming language designed specially for the teaching of Algorithmic Processes. In the first section two alternative approaches to the teaching of this first course in Computer Science are described. Next the authors state their basic hypothesis: that a programming language may be custom-designed for a given flow chart language with ensuing benefit. A flow chart language is chosen and described in detail with several examples of complete problems solved in flow chart form. Next the programming language is described and the problems solved in the previous section with flow charts are now coded in this programming language. Several simplifying features of the programming language are described. This programming language is now being utilized in a college classroom situation in order to evaluate its effectiveness as a teaching tool. A brief description of its current use and the means of its evaluation are presented in the last section.

## Dual Purpose of the First Course in Computer Science

The first course in Computer Science curriculums often is titled "Introduction to Computer Science"; sometimes "Introduction to Algorithmic Processes"; and perhaps sometimes "Introduction to Programming". The latter two titles both suggest slightly different shades of emphasis. Some schools attempt to emphasize the programming aspect of this first programming course--the goal being the greatest fluency possible in one programming language--usually FORTRAN. Other schools place primary emphasis instead of developing as firmly as possible good problem-solving habits, i.e., effective analysis of algorithmic processes. It should not be construed that these goals are necessarily mutually exclusive, however, there are a sufficient number of "Instructional Detours" that one must make in teaching the programming language such that a considerable fraction of the instructional hours available are spent off the main path. Instructional detours are required for some elements of the programming language which do not contribute to the algorithmic analysis, such as, formatted I/O, declaration statements, internal number representation, etc. One approach to alleviate this problem is to design a programming language which reduces the disparity between the algorithmic analysis and its representation in a programming language to a minimum.

---

This work was supported in part by ONR Contract N000-14-68-A-0151. (Project Themis at the University of Houston).

## A Hypothesis Concerning Flow-Charts and their Representation in a Programming Language

Most courses that emphasize algorithmic design use a flow-charting discipline to solidify the student conventionalization process. This is the case here at the University of Houston where the students are encouraged strongly to "solve" their problems completely with flow-charts before going to the computer. Thus within this framework the instructional detours, amount principally to the disparities between this flow-charting "language" and the programming language. If this disparity can be reduced to a minimum then the main theme of the course can be pursued more fully.

In order to illustrate and investigate this hypothesis a particular flow-charting language must be chosen from among the several available. Therefore, the flow-charting language used in the text Computer Science: A First Course was chosen for this purpose.<sup>1</sup> This particular flow-charting language and text were chosen for several reasons: first the flow-chart language is well defined and rigorously applied to many examples and; secondly the authors have considerable personal familiarity with the text since it has been used at the University of Houston for several years. The choice of this particular text and flow-chart language is not essential to the hypothesis--that given a flow-chart language, a programming language can be designed which reflects as closely as possible the structure and elements of flow-chart language thus eliminating spurious disparities. Indeed any other flow-charting language could have been alternately chosen. In the next section the flow-chart language will be defined and in the section immediately following that a simple programming language based on the flow-chart language will be presented.

## The Flow-Chart Language

Basically the flow-chart language consists of numerous types of numbered "boxes" connected by arrows with logical and/or arithmetic expressions within these boxes.

The basic symbols within the flow-charts consist of flow-chart variables, operators and numerals.

A flow-chart variable consists of an alphabetical character followed by from zero to five alphanumeric characters. Numerals consist of decimal digits which may or may not contain a decimal point (which has its usual meaning). Operators include the following:

---

<sup>1</sup>A.I. Forsythe and others, Computer Science: A First Course. New York: John Wiley & Sons, 1967.

Operator	Meaning
+	add
-	subtract
x	multiply
/	divide
←	assign
↔	interchange (two variables)
↑	exponentiation
=	equal
≠	not equal
<	less than
>	greater than
≤	less than or equal
≥	greater than or equal
"	quote
AND	and (logical)
OR	or (logical)

FIGURE 1: Flow-Chart operators and their meaning

Examples of valid expressions are: A, K25, A+B, X+X+2.5XY

Parenthesis may be used to imply the order of operations or to increase readability.

Subscripts may be appended to flow-chart variables. A subscript is distinguished from a flow-chart variable by the fact that it is printed in smaller type and one-half character below the line.

Functions are denoted by a flow-chart symbol followed by left and right parenthesis which contain (between them) the arguments to the function.

The shape of the boxes is very significant in this flow-charting language implying the use to which the expressions within the boxes will be put and the relation between boxes. The basic shapes of the boxes are illustrated in Figure 2.

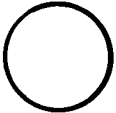


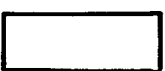
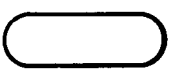
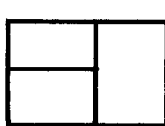

Shape	Function
	start or stop algorithm
	input
	output
	assignment
	decision
	repetition or iteration
	call on a "special" procedure

FIGURE 2: Flow-Chart boxes and their meanings

Three flow-chart examples are given in Figures 3, 4, and 5. The first, Figure 3, is the algorithm for finding the first value over 1,000 in the Fibonacci sequence. The second, Figure 4, is the Euclidean algorithm for the greatest common divisor, and the third, Figure 5, is the shuttle interchange sort algorithm. An examination of these flow charts will acquaint the reader with the flow-chart conventions.

A feature of this flow-chart language not emphasized in the text but relied on by the author in the design of the programming language is the fact that flow chart boxes are always numbered.

### The Programming Language

Figures 7, 8, and 9 are examples of the programming language as a representation of the flow charts given in Figures 3, 4, and 5, respectively. As the following description of the programming language is read it may be useful to consult these figures for concrete examples.

Basically the programming language consists of statements each of which must be preceded by a label and followed by one label or by two labels separated by a comma. A label consists of the letters "BOX" followed by one or more digits. A vertical bar indicates both the beginning and the end of a statement. The vertical bars are intended to represent the "edge" of a flow chart box to the student. To increase this suggestive quality, decision statements are enclosed in left and right parenthesis to suggest the curved sides of the decision box and call statements are enclosed by two vertical bars rather than one. Also since the flow chart language allows for several assignment statements within an assignment box, the programming language allows for several distinct assignment statements to be written as one statement; each assignment must be separated from the next by a semicolon (See Figure 8 for an example). Statements are composed of program variables and operators and numerals. Program variables and numerals are defined identically to flow-chart variables and numerals, respectively. The same symbols are used for operators with the following differences.

Flow-Chart Symbols	Program Symbol	Meaning
x	*	Multiply
↑	%	Exponentiation
≠	≠ = (two symbols)	Not equal
≤	< = (two symbols)	Less than or equal
≥	> = (two symbols)	Greater than or equal

FIGURE 6: Operator Discrepancy Between the Flow-chart language and the programming language

These changes in the operators were necessitated by limitations on the availability of characters imposed upon us by the line printer.

Subscripts are treated similarly to FORTRAN, i.e., the subscripts are enclosed by left and right parenthesis. Functions may be distinguished from subscripted variables due to the fact that their argument lists are enclosed by left and right

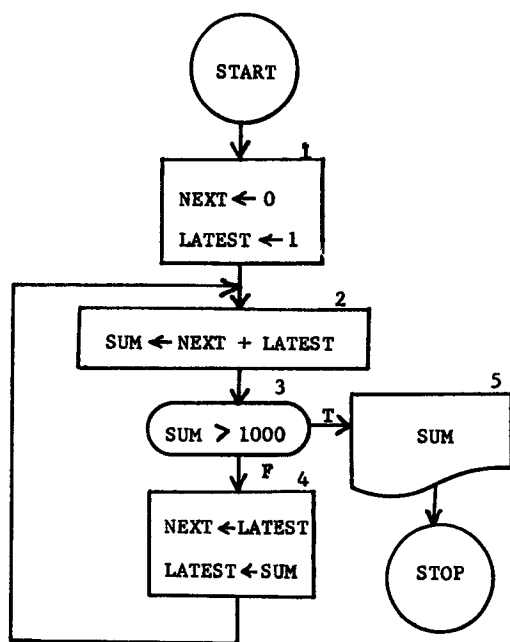


FIGURE 3

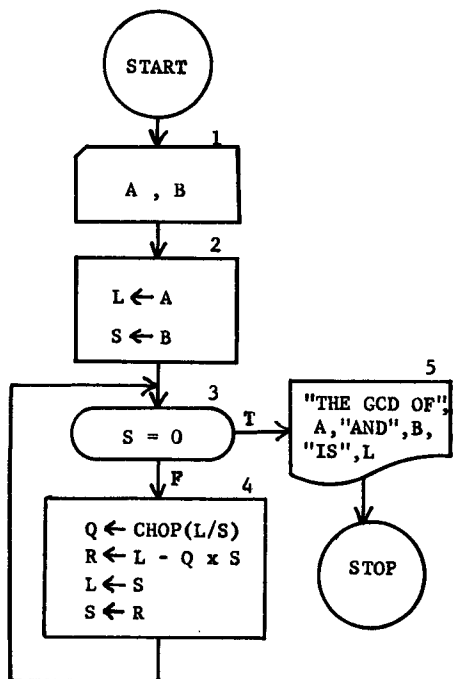


FIGURE 4

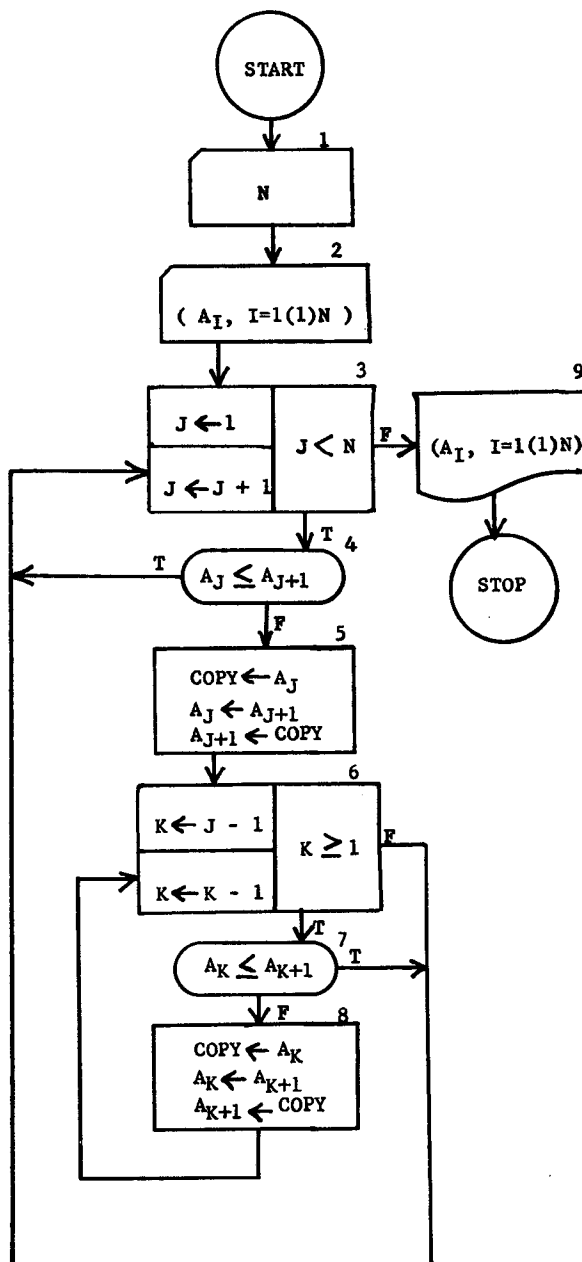


FIGURE 5

```

***** FLO-TRAN PROCESSOR ** VERSION 1 ***
***** FLO-TRAN PROCESSOR ** VERSION 1 ***
STUDENT NAME :LARRY C. EVERSOLE,INSTRUCTOR STUDENT ID NO. :001
*****FLO-TRAN TRANSLATION BEGINS *****
START BOX1
BOX1 I NEXT <- 0 ; LATEST <- 1 / BOX2
BOX2 I SUM <- LATEST + NEXT I BOX3
BOX3 ( SUM > 1000 ) BOX5,BOX4
BOX4 I NEXT <- LATEST ; LATEST <- SUM I BOX2
BOX5 I OUTPUT SUM I STOP
**
TRANSLATION COMPLETE
***** EXECUTION BEGINS *****
1.597,+03
NORMAL PROGRAM TERMINATION

```

FIGURE 7

```

***** FLO-TRAN PROCESSOR ** VERSION 1 ***
***** FLO-TRAN PROCESSOR ** VERSION 1 ***
STUDENT NAME : WM. MAX IVEY STUDENT ID NO. :000
*****FLO-TRAN TRANSLATION BEGINS *****
START BOX1
BOX1 I INPUT A,B I BOX2
BOX2 I L <- A ; S <- B I BOX3
BOX3 ( S = 0 ) BOX5,BOX4
BOX4 IQ <- CHOP<L/S> ; R <- L - Q * S ; L <- S ; S <- R I BOX3
BOX5 I OUTPUT THE GCD OFR,A,RANDR,B,RISB,L I STOP
**
TRANSLATION COMPLETE
***** EXECUTION BEGINS *****

THE GCD OF
9.430,+02
AND
4.370,+02
IS
2.300,+01
NORMAL PROGRAM TERMINATION

```

FIGURE 8

```

***** FLO-TRAN PROCESSOR ** VERSION 1 ***
***** FLO-TRAN PROCESSOR ** VERSION 1 ***
STUDENT NAME :LARRY C. EVERSOLE,INSTRUCTOR STUDENT ID NO. :001
*****FLO-TRAN TRANSLATION BEGINS *****
START BOX1
BOX1 I INPUT N I BOX2
BOX2 I INPUT (A(I),I=1(1)N) I BOX3I
BOX3I I J <- 1 I BOX3T
BOX3T ( J < N ) BOX4,BOX9
BOX3K I J <- J + 1 I BOX3T
BOX4 ( A(J) <= A(J+1) ) BOX3K,BOX5
BOX5 I COPY <- A(J) ; A(J) <- A(J+1) ; A(J+1) <- COPY I BOX6I
BOX6I I K <- J - 1 I BOX6T
BOX6T ( K >= 1 ) BOX7,BOX3K
BOX6K I K <- K - 1 I BOX6T
BOX7 ( A(K) <= A(K+1) ) BOX3K,BOX8
BOX8 I COPY <- A(K) ; A(K) <- A(K+1) ; A(K+1) <- COPY I BOX6K
BOX9 I OUTPUT (A(I),I=1(1)N) I STOP
**
TRANSLATION COMPLETE
***** EXECUTION BEGINS *****
-5.386,+02 -6.450,+00 3.457,+00 4.500,+01 8.770,+01
2.450,+02 7.533,+02 2.549,+03 3.524,+03 1.438,+04
NORMAL PROGRAM TERMINATION

```

FIGURE 9

pointed brackets (" $<$ " and " $>$ "). Read and write statements are indicated by the occurrence of the word INPUT or OUTPUT, respectively immediately after the beginning vertical bar. Conditional expression (no arrows present) are surrounded by parenthesis and have two labels following the statement. The repetition box which is actually composed of three connected boxes has been "split-up" into three statements each of which corresponds to one of the three parts of the repetition box. The initialization box, test box, and incrementation box are denoted by appending the letter I, T, and K, respectively to the end of the label of the statement (Refer to Figure 9 for several examples of iteration boxes). Three examples of the programming language are given here to help clarify any further points. Figure 7 is the Fibonacci sequence algorithm. Figure 8 is the program to find the greatest common divisor using the Euclidean (division) algorithm and Figure 9 is the shuttle interchange sort.

#### Simplifying Features of the Language

Among the simplifying features of the language are the following:

1. The programming language contains only those statements which are found in the flow-chart language. It has no declaration statements such as DIMENSION, INTEGER, etc. such as are found in FORTRAN and other high level languages. No GO TO statements are necessary.
2. The flow-chart language does not contain statement types which have no correspondence in the programming language, although the notation in some types, i.e., read and write, has been changed slightly.
3. Numerals are written as in arithmetic without consideration to their internal (integer or real) representation.
4. I/O has been simplified. There is nothing in this language which corresponds to a FORMAT statement in FORTRAN, an item which is not represented in a flow-chart.
5. Whenever possible the same symbols are used to denote the same operations in both the flow-chart language and the programming language.

#### Limitations of the Programming Language

The reader should remember that the programming language described herein is designed strictly for instructional use and not for any type of commercial applications programming. When evaluated in comparison with FORTRAN, ALGOL, COBOL, PL/1 or any other production language it has many limitations.

The programming language described here is particularly limited in its input and output capabilities. It can only input numbers and output literal character strings and numbers. The user has no control over the appearance of the output produced by his programs.

The programming language does not have the capability to represent data in different ways such as integer, decimal, real (floating), complex, double precision, or character form. More over it does not have the capability of representing structural relationships between data items such as does COBOL, PL/1 and some implementations of ALGOL.

#### Evaluation of the Programming Language: An Experiment

An experiment is being conducted at the University of Houston during the current semester to evaluate the effectiveness of the programming language in the teaching of Algorithmic processes. Two classes of students registered for Computer Science 141 (Introduction to Algorithmic Processes) are being used in the experiment. Both classes meet on Mondays and Wednesdays, one from 5:00 to 7:00 p.m. and the other from 7:00 to 9:00 p.m. A recently administered test demonstrated that the classes score about equally well on purely flow-charting problems. The first class is now being taught a standard high-level language (FORTRAN) and they will use this language to represent all of their flow-chart solutions for the remainder of the semester. The second class will be taught the programming language described within this paper and they will use it to represent all of their flow-chart solutions. The same instructor will teach both classes (Larry C. Eversole).

Every attempt is being made to see that each class receives about the same number of instructional hours on the programming language which it is being taught. Each class has identical opportunities for access to the UNIVAC 1108 upon which both classes will run their computer problems, since language processors for the two languages are both available at all times during the day.

During the semester and at its conclusion data will be gathered in an attempt to evaluate the performances of the two groups. Data which will be taken concerns the following:

1. The number of computer runs each student makes during the semester.
2. The students' scores on midsemester and final examinations.
3. The average number of runs required to successfully complete each assigned computer problem.
4. The percentage of successfully completed computer assignments, per assignment.
5. Students personal opinions (from a questionnaire).
6. Time spent debugging.

Results of this evaluation will be available sometime during the summer and any interested parties are urged to write the authors for information. Any suggestions would also be appreciated.

---

The authors wish to thank Professor R. A. Sibley, Jr., University of Houston, for his valuable suggestions and assistance throughout this research project.