INSTRUCT

An Introductory Conversational Programming Language

A Thesis
Presented to
The Faculty Of Graduate Studies And Research
The University Of Manitoba

In Partial Fulfilment
Of The Requirements For The Degree
Master Of Science
In The Department For Computer Studies

By
Kenneth B. Bileski
May 1970

Abstract

The advantages and disadvantages of the use of several existing conversational languages in an introductory programming course are discussed, and the basic requirements of an introductory conversational programming language are considered. INSTRUCT (an introductory conversational language) is proposed to fulfill these requirements and its design goals are stated.

The syntax and semantics of INSTRUCT are specified and the details of its implementation on an IBM 360/65 are outlined. Example programs illustrating the use and special features of the language are contained in the various chapters and in the appendices.

Acknowledgements

I would like to express my sincere thanks to Professor S. R. Clark, my supervisor, for his advice, and assistance, which have been essential in the preparation of this thesis.

My gratitude is also expressed to Mr. D. Costin for his comments and criticisms.

Finally, I wish to extend special thanks to Mrs. M. Thomson for all her patience, help, and for typing this thesis.

Table Of Contents

Table Of Figures

CHAPTER I

Introductory Conversational Programming Languages

1.1 Introduction.

At present many high schools in the United States and Canada are offering courses in programming languages. One of the major problems of introducing a programming course into a high school is the implementation of a method for processing student jobs.

At the present time in most universities student jobs are processed using batch processing methods. The student punches his program onto cards, submits the deck, and waits for his output to come back. Upon receiving his output the student decides whether or not corrections are required. If they are required the corrections are made and the deck is resubmitted.

However, the rapid advances in computing systems have made it possible for users to interact with a computing system on a conversational basis, i.e., the user can input information directly into the computer,

the computer feeds results back almost immediately, and the user's next input of information can be influenced by results produced from previous inputs. In this case there is a continued interaction between programmer and the computer.

1.2 Batch Processing Methods For Teaching.

This is the usual method of processing student jobs at most institutions where programming languages are taught. From the author's experience of demonstrating a first programming language course (WATFOR [1]), where batch processing methods are used, several advantages and disadvantages become apparent.

1.2.1 Advantages of Batch Processing.

A large number of student jobs can be processed in a short time, at a specified time if desired, thus freeing the computer for other uses.

Students are forced to be reasonably careful in the preparation of their programs as a simple error, for example a misspelled variable name, causing a run to fail may mean the student has waited X number of hours for his output to come back, and then found he had attained nothing towards his goal of producing a running program.

Most computer centres use batch processing methods thus, at the end of an introductory course, the student is accustomed to programming and debugging in this type of processing enviornment.

1.2.2 Disadvantages Of Batch Processing.

If the time between submission of a student program and the returning of the output is lengthy, it slows up the student's progress in learning the language.

Quite often a student making corrections is racing the clock so he can catch the "next run". This can cause a student to make corrections on the basis of quick guesses about what is wrong.

Any inclination the student has to experiment with different features of the language may be limited by the time he has to wait from submitting a deck to the time he gets his output back (TURNAROUND). This is especially true if there is a deadline on the time a student has to produce a working program. The result is many students learn a subset of a language and program within the bounds of what they are familiar with.

In any batch processing environment there is always the problem of the occasional student deck or output being lost.

If a student loses runs because of careless errors such as keypunching errors, incorrect positioning of job cards, etc., besides being frustrating, it can also waste a lot of computer time.

Batch processing can cause the finding of logic errors to extend over a fairly long period of time if the student requires several runs to accumulate enough information to lead him to the cause of the error.

1.3 Conversational Processing Methods For Teaching.

Though the author's experience with a conversational programming language is restricted to a few small trial APL [2] (A PROGRAMMING LANGUAGE) programs several advantages of using a conversational programming language for teaching become obvious. However some disadvantages, though they are less severe than those associated with batch processing, are apparent also.

1.3.1. Advantages Of Conversational Processing.

The student interacts directly with the computer and vice versa. This direct interaction speeds up the learning of a new language tremendously, and usually increases the student's interest.

The correction of syntax errors is much easier and faster than in batch processing. The student can now spend more time developing the logic and techniques to effectively handle the assigned programming problems.

It is possible for a student to have several trial runs of his program in a period of time that is determined mainly by the time it takes the student to modify his program. This encourages experimentation with unfamilar features of the language.

The time involved in finding logic errors is greatly decreased, especially if adequate debug facilities are available.

1.3.2 Disadvantages Of Conversational Processing.

The major disadvantage will be in handling volume. The typing in of long programs would be

tedious and time consuming unless a facility for storing a program, which would be called back whenever the user wished to use or modify it, was available.

The inputting or outputting of large amounts of data would be awkward. This disadvantage could be circumvented by having an option whereby the input or output of data was directed through a reader/printer once the program was working. However for most student jobs the amount of data input or output will be small.

A problem could arise in providing enough terminals for the large number of users usually associated with an introductory course.

A fairly large computing system comparable in size to an IBM system 360/50 [9] or larger would be required to handle a large number of programs being run in conversational mode.

1.4 Some Existing Conversational Languages.

A limited number of conversational programming languages have come into use, and it is almost certain

that the use of conversational programming languages is going to increase rapidly in the near future.

This will be due to the distinct advantages for the user in this type of programming, the rapid improvement of software, and the general increasingly powerful computing systems coming onto the market which will quite likely overcome most of the disadvantages associated with conversational programming.

The conversational programming languages in existence today, however, would not be very suitable for an introductory course in programming at the high school level, as a short look at two of the more popular conversational languages, APL [2] and BASIC [3], shows.

1.4.1. APL As An Introductory Conversational Programming Language.

APL is a high level conversational language (a high level language is one which is more English-like or mathematically orientated than the basic machine language) that can be very effective when used by an experienced programmer. However it was not designed to

be an introductory conversational language, but rather as a language containing very powerful functions which would reduce the amount of coding necessary to produce a working program.

1.4.1.1 Advantages of APL.

The student would be familiar with at least a subset of a very powerful programming language at the end of the course.

1.4.1.2 Disadvantages of APL.

The major difficulty with the use of APL as an introductory conversational programming language would be its considerable complexity. Though any student who continues his studies in the field of Computer Science will eventually have to learn languages of equal or even greater complexity, use of such a complex language in an introductory course could easily overwhelm and/or discourage the average high school student.

Since APL is a high level language the syntax of the language completely disguises the basic machine

instructions. The following two examples illustrate this:

1) Assigning values to an array.

   The APL statement is:

   ⍞←M←2 3⍴4 6 12 4 2 8

   The above statement places the values 4 6 12 in the 2 x 3 M array.

2) Concatenating the components of vector B to vector A.

   H←"QUARTER"

   G←"BACK"

   H,G

   The result of the above APL statement is the vector "QUARTERBACK".

Presumably in an introductory course in computer programming it would be desirable to demonstrate how the computer operates using the language being taught. With APL this would be very difficult, because the language is of such a high level.

Also the language is highly orientated towards mathematics as shown by the functions available, the mathematical type of syntax, and the symbolism. It is doubtful if the average high school student would have

the mathematical background to enable him to quickly grasp the terminology involved in APL. An example of the mathematical type of syntax and symbolism used in APL follows:

1) Consider the APL statement
6 . 9 8 7 + 2 3 2⍴⍳2 1 2

⍳ - this function generates a vector of all the possible indices in ascending order of a vector with ⍵ components, starting with the index origin.
Therefore ⍳12 produces a vector containing
1 2 3 4 5 6 7 8 9 10 11 12.

⍴ - produces an array.
Therefore 2 3 2 ⍴ means produce a 2 x 3 x 2 array using the vector produced by ⍳1 2.

To each element of the 2 x 3 x 2 array produced, 6 . 9 8 7 is added.

The results produced appear as follows:

7.987	8.987
9.987	10.987
11.987	12.987

13.987 14.987

15.987 16.987

17.987 18.987

From the preceding examples it is apparent that APL would not fulfill one of the basic requirements of an introductory language, I.E.: simplicity of syntax, as defined in IPLAN [4] .

1.4.2 Basic As An Introductory Conversational Programming Language.

The programming language BASIC which may or may not be implemented as a conversational language was designed to be a comprehensive introduction to the art of computer programming. This language is rapidly gaining acceptance as a general-purpose computer language, particularly in the Time-Sharing System for the PDP-8/1 and PDP-8 computers [5] (TSS/8).

1.4.2.1 Advantages Of BASIC.

The course would be an excellent introduction for a student who plans to eventually learn FORTRAN [6] , or ALGOL [7] , or any other high level language (according to the authors).

The language is powerful enough so that the student can use it to solve fairly complex problems.

1.4.2.2 Disadvantages of BASIC.

Explicit declarations of data types are not a feature of the language. Thus the student is not working with the concepts of different types, range, and accuracy as related to real and integer numbers.

The language is of too high a level to demonstrate the operation of a computer. The arithmetic instruction set is not in a one-to-one relationship with the basic machine instructions, as the following BASIC arithmetic assignment statement shows.

```
LET  S = (T+2*M)/3
```

In this statement the basic machine instructions of multiplication, division, addition, and the storing of the results in S are all collected into one statement.

The concept of the meaning of = sign in BASIC may be difficult for some students.

The presence of both the function call and array reference features may present problems. For

example the reason why SIN(X) calls a function to calculate the sine of X, and why S(X) is a reference to the Xth element of the vector S may be difficult for students to comprehend.

The syntax error messages in BASIC are inadequate. The instructions that are at fault are flagged but comprehensive error diagnostics are not given.

The debugging facilities are also inadequate. The tracing of the flow of a program and the dumping out of intermediate results all have to be done by using PRINT statements supplied by the programmer.

1.5 The Introductory Conversational Language INSTRUCT.

INSTRUCT attempts to fulfill the need for a conversational teaching language which has the basic requirements of an introductory programming language.

1.5.1 Origin Of The Syntax And Semantics Of INSTRUCT.

The syntax and semantics of INSTRUCT originate from the introductory programming language IPLAN which was designed and implemented for use in a batch processing environment at the University of Manitoba [4]. INSTRUCT is a modified version of IPLAN which is designed for student use in a conversational processing environment.

1.5.2 Basic Requirements Of An Introductory Conversational Language.

The basic requirements that IPLAN was designed to meet and which are also felt by the author to be necessary for an introductory conversational programming language are:

(I) Simple Syntax.

(II) Simple but powerful input-output.

(III) Comprehensive error diagnostics.

(IV) Adequate debug facilities.

(V) Explicit data types.

(VI) An arithmetic instruction set which is in a one-to-one relationship with basic machine instructions.

(VII) Conditional and unconditional branch statements (using statement labels).

(VIII) Subscripting and loop facilities.

Requirements that the author feels necessary to be added for an introductory conversational language are:

(IX) Number of statements executed as a measure of program efficiency.

(X) Simple, explicit commands for conversationally modifying a program during compilation or before the start of execution.

(XI) Any conversational requests output by the compiler to the user should be simple, easy to understand, and should define the various acceptable responses the user can input.

(XII) The user should have the facility of terminating an executing program. This feature would be especially useful when the program is obviously producing incorrect results and the user wishes to make corrections and to execute the program again, without waiting for the program to stop executing on its own.

As experience is gained in the use of conversational programming languages for teaching it may be found that other features should be added, and perhaps that some features should be dropped. The requirements stated above are only a proposal, and alterations are expected with increasing experience in this area.

1.5.3 INSTRUCT DESIGN GOALS.

The design goals are:

(I) Ease of Learning.

The techniques required to conversationally program using INSTRUCT from an IBM 2260 DISPLAY STATION [8] (hereafter abbreviated as CRT) terminal must be easy to learn. A CRT terminal was chosen instead of an on-line typewriter terminal because a CRT would be faster at handling output, and would also eliminate the usage of paper as the recording medium for input and output (unless an option was available which allowed the user to direct his input, or output, or both to a printer). Each INSTRUCT statement occupies an area of seventy-five characters on a CRT. The START SYMBOL [8] is always moved to the beginning of the next area to be used after a statement has been entered. The source statement and the input data will be free format.

(II) Comprehensive checking of non-logic errors.

If an error is discovered in an input source statement the user is notified immediately with a full diagnostic error message. The statement in error is automatically deleted from the user's program, and the next statement entered is assumed to be its replacement.

In the case of a non-terminal execution error the user will be notified what kind of error occurred, which statement it occurred in, and what corrective action has been taken. The user is given the option of allowing the program to continue to execute, or of stopping execution.

In the case of a terminal execution error the user will be notified what kind of error occurred, which statement it occurred in, and that execution was terminated. The user now has the option of modifying his program and executing it again, or of terminating the job.

(III) Comprehensive facilities for tracing logic errors.

Easy to use debug facilities are provided for tracing the execution of statements, and the type and value of the accumulator as each statement is executed.

(IV) Efficient implementation.

Fast compilation and execution of INSTRUCT programs in order to keep the response time to a minimum as far as the INSTRUCT compiler is concerned. Naturally the response time of a INSTRUCT user will depend upon the environment in which INSTRUCT operates.