To: P. S. Dauber 22-242 Yorktown Heights

TO:

FOR YOUR INFO | COMMENTS | FILE

965 - IBM - 04

Date: February 15, 1974

FEB 19 1974

Name & Tie Extn: P. C. Goldberg/862-2136

Title/Dept. Name: Manager/Automatic Programming

Mail Zip City, State: 27-205/Yorktown Heights, New York

or U.S. mail address: P.O. Box 218

PLS. HANDLE | RETURN | DISCUSS WITH ME

FROM:

Subject: John Backus' Remarks on Red Languages and Reducing the Cost of Programming

Reference:

I think that we all agree with John that reducing the cost of programming is a critical issue. I also agree that the production of languages of the PL/I genre, even if considerably more versatile and easier to use than PL/I, will not essentially change the current situation. I, too, believe that the APL experience should be taken seriously and that one of the critical issues in the area of programming language design is the development of appropriate aggregate operations for particular application areas. I am not, however, convinced that we ought altogether to dispense with the notion of an explicit store and an assignment operator.

John's memo raises three issues. First, we must discuss what kinds of programs we need to produce and where the programming problems are. Second, it is important to understand what formalisms are available to perhaps guide us in the construction of languages to meet these needs. Finally, the question arises as to what serious work the Yorktown Computer Science Department is doing to decrease the cost of programming.

The cost of programming must be reduced not only for professional programmers writing large systems, but also for casual users of computing machines. We must create languages with great expressivity but which also make possible easy debugging and modification of programs. One of the failings of APL is that the intent of programs is difficult to extract, and so use by persons other than the creator is made difficult. More specifically we must create languages which are more problem oriented and also languages to deal with large data bases in terminal oriented situations. Language development that concentrates on the internal computational power and ignores these issues is misguided.

There are a number of radical, i.e. non-Von Neumann, formalisms which have been advanced for use as a basis for programming language instruction. There are, of course, John's Closed Applicative Languages, but there are also the lambda calculus model and data flow models, each of which is attractive in certain ways. My own feeling is that it is important to study these

models in order to get a better appreciation of where their power lies and thus how they can best be exploited. Such studies have been carried on in Computer Science by, for example, Paul Kosinski in studying data flow models and by Bill Burge in his work with lambda calculus based languages.

However useful these formalisms are, however, I do not believe that programming language development should be guided by rigid adherence to one or another of them. Rather I think that the intended problem area for the language should be well understood and that various aspects of these formalisms should be introduced into a language in such a way as to solve the problems of programming for that environment. Thus, for example, I believe that it is possible to introduce the notion of function producing functions, which is the mainstay of the lambda calculus, into a language with assignment in which the basic operations are aggregate operations -- without developing a language too unwieldy for practical use. In a similar vein the Application Generation Systems group has developed a language which combines, at appropriate levels, data flow notions with notions of transformations on sets of objects together with notions of assignment to produce a language appropriate for specifying business applications.

Finally, I would like to suggest that not all work aimed at reducing the cost of programming needs to be concerned with the development of programming languages. I think that our work in debugging systems, our work in terminal interfaces, indeed work in such areas as proof of program correctness and general deductive systems, contribute to that end.

P. C. Goldberg

cyr