APL\360 HISTORY

by

Adin Falkoff
Manager of the APL Project

I am very pleased to see so many faces here that I know, and I am even more pleased to see so many that I don't know. This is a nice turnout. I am glad it happened.

The program says that this is going to be the history and status of APL\360. It will mostly be the history of APL, and I think that is reasonable because APL\360 is part of the history. I suspect that there will be many other APL\something-or-others before too long.

Most of you know that the whole thing was started by Ken Iverson. You may not know that he started it as far back as 1957 when he was teaching at Harvard, and there were certain motivating factors then which it seems have consistently threaded through the rest of the history of APL until now, and I would like to mention what I consider to be the three main lines of continuity, as it were.

First of all, there was the notion of communication; as Dr. Dearing mentioned, communication is vital in so many phases of our activities. The original problem was, in the process of writing a book, to communicate something about how sorting is done on machines and, then, at about the same time, to communicate to students something about linear programming. There has never been in Ken's thinking or in the thinking of most of us who work with him any sharp distinction between communicating with a machine or communicating with people, or for that matter, communicating with yourself (just to be sure that nothing is left out). So we have that one very significant factor.

The second point that I would like to make is that the motivation for the development of APL and for the development of implementations of it have never been for the purpose of developing a language per se. The motivation has always come from applying what we already had to a new problem or exploring some old problem in further depth. This has consistently led not only to additions to the language but to simplifications and unifications, which I hope will become evident.

The third principle which I think has been with us since the earliest times is the idea that programming and mathematics are inseparable. Mathematics, as it is normally taught and conceived of, tends to be somewhat static; it deals with structures. But behind the static aspect of it there is the notion of transformations on the structures and processes that take place to either build them or exploit them in some way. The attention of most mathematicians

so far has been directed not at these dynamic aspects of it, but at the more static aspect, and I think that programming, or if you like, algorithms, represent that part, and the advent of computers has directed people's attention more to this dynamic aspect of mathematics.

With those three notions in mind then, I would now like to follow two threads. I don't know if I will be following them concurrently or in some mixed up way, but the two threads that I would like to follow are the development of the language--changes that have taken place since the earliest publications, and the history of developments of things like new systems and implementations and things of that sort. I confess to some doubt as to the strict scholarly knowledge of some of the very early history. I know that Ken Iverson gave papers before his book was published. (I am not sure; I believe there was one published in the proceedings of a London conference on something.) Certainly the first major publication which outlined the full potential of the language was the book published in 1962 called A Programming Language. The next major publication which modified the language came about two years later, at the end of 1964, when the paper on a formal description of the System/360 was published, and the next major modification came about two years after then, in 1966, when the first manual for APL\360 was printed. I would like to trace for you the changes in the language that took place over those three steps.

The major trend that I see in the change from the book to the formal description was a unification of the treatment of scalar operators. There was also at that time a recognition of the value of dispensing completely with operator precedents and letting the order of execution come from right to left. We noted particularly at that time the advantages in reduction in making that rule (you may have noticed that in the book the reduction goes from left to right), and there was considerable debate among us at that time about changing the rule and I think that what really swung the balance was the recognition that this would give far more interesting mathematical functions than the other; namely, that reduction with things like minus and divide gave you alternating sums and alternating products, which were not easily achieved otherwise, whereas the left to right was kind of a dull thing, just the first element minus the sum of the others. And so this, plus the trend towards uniformity, sort of forced that particular development.

Another detail at that time was the removal of the ... (I was about to say the removal of the elision, which is carrying things a little too far). In any case, some elisions were allowed in the language, as defined in the book, and particularly multiplication of a scalar by a vector could be written in a variety of ways. One of these simply had the two characters juxaposed, and it was decided at the time of the formal description that, again as part of the notion of unifying the treatment of the scalar operators, that would have to go. At the same time, this pointed the direction for allowing scalars to interact with vectors with an operator which, of course, you recognize as a kind of a major extention of the rules of linear algebra. Again, as part of this trend towards uni-

fication, the operators that we know now as the "min" and the "max" in the book were described as "minimum selectors" and "maximum selectors" and had a fairly involved, albeit useful, definition. You couldn't use them as simply in reduction to find the largest element of an array or a vector. It was necessary to modify them by a selection vector first, and they had some other quirk, as well. Well, they were modified at that time, although in checking this morning we discovered that there is no overt mention of them in that particular publication. Ken and I are convinced that is what we had in mind at that time. It is just that if you are dealing with something that involves description of a machine and the largest value you ever get is one using a maximum over it, it isn't terribly useful.

Another notational step that was taken at that time, and this has to do with cleaning up the syntax, was the dropping of the closing symbol for absolute value floor and ceiling. I have observed that in certain ivory towers of pure mathematics they have adopted the floor and ceiling symbols still with the pair about them, and I estimate that in roughly another decade they will get around to dropping the second.

Another thing that was foreshadowed by the book was the description of concurrent operations, but this was somewhat used in greater detail and more explicitly in the formal description. We, therefore, did introduce at that time a mechanism for handling concurrency or what we might call timing questions. Possibly the one new thing that was added at the time of the formal description was in many ways the most facinating and certainly the most powerful of all the operators and this was necessary because we were dealing with hardware which in some respects was unpredictable. We introduced the question mark which generated a random zero or one which sort of goes to show you that, where you expect to be on solid ground the most, you end up having introduced the indeterminent thing. So, at the end of 1964, when that paper was published, the language had changed some, but in the direction of simplification and unification.

It was about that time that we began to think rather seriously of implementations. We came in contact at that time with John Lawrence who was the editor of the IBM Systems Journal where this paper was published, and he encouraged us very strongly in the work that we were doing; he was about to undertake the beginning of a program with Science Research Associates and solicited our interest in that. This, coupled with some long abiding interests in having an implementation, led us to the design of the typing element which went through three stages of development to where it is now. We took a misstep, then the people who build the elements took a misstep, and then we got together. We have some interesting type balls lying around in the backs of drawers which have sort of partial characters on them.

I think I can recognize ... four major trends. (I see that I am going through this thing more or less segregated; that is to say, I am going through the technical history. I will have to leave the

other part of it, I guess, to treat separately.) I can identify four major trends demonstrated by the change to the implementation. Certainly, of course, one of the most obvious is the linearization of the language and that, however, is, well, let me say that while it was certainly forced by the desire to use a typewriter and to use it simply, the effect of it was in general good, and let me give you some examples. In the matter of using sub-scripts and super-scripts for indexing, if you are going to do that, you are fairly well limited to two dimensions; if you want to sneak around the back side of characters you can get four dimensions, but if you want to go any higher you are more or less stuck; so, by linearizing that operation it is possible now to generalize to arbitrarily large numbers of dimensions without changing the notation. Another thing is the good old notion of exponentiation where again a super-script is used, and linearizing that forces you to introduce an explicit operator symbol. Having done that, you now have consistency on all of the common arithmetic operators. You have another nice example of a non-permutative operator for teaching students, and, in fact, if you are going to stick with conventional precedence arrangements, it is one which associates from right-to-left rather than left-to-right, as do others in the elementary treatment of arithmetic.

Finally, the third use for super-scripts and sub-scripts comes from the desire to identify different versions of the same variable, like, successive derivatives or whatnot. This is forced on conventional algebraic notation by the fact that no symbol may consist of more than one character, and once you allow multi-character symbols you are free to dispense with those super-scripts. Of course you cannot allow multi-character symbols unless you did not allow elisions because, if you allow elisions, you don't know when your multi-character symbol is a multiplication or whatever the operation might be, and when it is a single symbol, so you see these things all hang together. So that was linearization as one major trend, and part of that was generalization to multi-dimensional arrays which came about in reference to the indexing, but was also in the extensions of other operations; for example, the allowing of scalars to interact with arrays in any rank and expand too on an element-to-element basis. Also, the extensions of the inner and outer product to their present state of generality.

The design of the type ball (type element) forced another kind of discipline on us which was only latent up to that point, and that was the economization in the use of symbols; and there are two ways in which this economy was manifested. One is that hardly any symbol does not have both monadic and the dyatic interpretation. Whereas formerly this was only true of the minus sign, it is now true of almost anything; e.g., the divide which has the obvious interpretation. The other way in which we economized was by the use of overstrikes; e.g., in the transpose or in the factorial. We have come to regard that as something of a principle. Another significant trend at that time, as I see it, was the simplification of the syntax. In the book, parenthesis are used in two ways, possibly in three, depending upon how you want to regard it. They are used

to determine association; they are used to append modifiers to certain operators like the epsilon (if you recall it was the $\epsilon(N)$ which gave the dimension of epsilon); and they were also used with the relations as a kind of a carry-over from some earlier influences that Ken met in his format studies. (You know, I had a rehearsal for this. There is a gentleman sitting in the second row here who interviewed us yesterday; and maybe I will put in a plug. There is a new magazine called Computer Decisions, and apparently we are to be featured in the first issue that is coming in August. The gentleman, Mr. Bairstow. In any case, he did come by yesterday and we had an opportunity to go over some of this material, and I learned a few things about Ken's earlier thought processes that I wasn't entirely aware of before.) In any case, with the production of implementations, we did simplify the syntax and parenthesis; for example, now they are used in only one way--that is to determine association, and a few other things have been cleaned up, and I must admit some of them with a little pain. For example, in the formal description System/360 ... we took unto ourselves the capability of defining a multiplicity of variables by putting all the names to the left of the arrow and putting the values to the right, and when we came to an implementation where somebody had to write a syntax analyzer and things of that sort, we found this wasn't quite as clean as it might have been, and so for the moment we had to give it up. Let me assure you, although even out of context, it has got to come back somehow.

Another small thing which I doubt many people will miss is that we abandoned the idea of using a pictorial symbol for special matrices, having discovered somewhat earlier that outer products gave these things with considerable convenience. I think another thing that came in with the implementation was the generalization of the operations represented by Rho. First of all is the monadic Rho which gives you the shape or size of an object and that replaced two earlier symbols, Mu and Nu, which you can see were good enough for matrices but couldn't go any further. For one thing, I don't know any other Greek letters. But Rho answered the problem and gave them all to us. And once again by "here was a symbol; what can we do with the left side of it?", and naturally if used by itself it gives you the size, used with the left argument it most certainly should give you something that has a size, and so we had the re-structuring operator born that way. That replaced a rather awkward and specialized kind of expansion that you will find used in the earlier publications.

Another thing that the implementation did for us, it forced us to consider the whole notion of function definition and introduced more formalization in that, in particular the idea of a function header with a syntax that could be defined. So that, then, sort of summarizes, as I see it, the development of the language, and I think that it is fair to say that all of the developments have been towards simplification and generalization.

There has been, aside from what Herb Hellerman calls "Penmanship", very little change in most of the operations; that is to say, there

is a startling difference in appearance between, say a program written as we did in 1964 and as we do it now. But if you know one, you can read the other with very little difficulty. Very little has been rendered obsolete in the course of this. One thing that has been, and gave us some pain, (I don't know if it gives you any pain) and that was the removal of the alpha and omega from the list of operators; but I don't think the released systems had those, so you never missed them. We had them and it was a bit of a jar to loose them. But they are gone now.

Let me now briefly review some other aspects of the history; I guess I have touched on a few of them. I mentioned that one of the three major threads that I see going through the entire development was the emphasis on applications as pointing the way towards language development, rather than seeking to develop the language for the sake of itself. Two applications, well one application, that was clearly present from the earliest days of Ken's work was the application of teaching, and there again we come to the matter of communication. The application in teaching, as all of you can attest, is now stronger than ever. It has gone through a number of phases. There was the original use that Ken put it to. Then there was the time that he started teaching at the Systems Research Institute, and then I took over there, and we used APL for teaching various aspects of programming and system design. There was the resurgence or emphasis on the use of teaching when John Lawrence and the SRA business came up, and we very deliberately, at that time, set out to see how we could use such a facility for teaching a variety of things; and then, of course, there was Ken's work in the secondary schools which led to his second book, not counting the third one which was jointly authored (that was the second perhaps), but anyway the book on the algorithms - "Elementary Functions." So, teaching has been a major application area which has led to a number of interesting developments.

The other major application area has been in system design. First, it is impossible to divorce any formalization of a process from the notion of designing a system once you accept the definition which I use, of a system which is a collection of programs; but in any case I think certainly when we started doing the formal description and earlier when Ken was working on things like the description of the 7090 which went into his book and my earlier work on the parallel search memories, the use of the language for describing systems was then a conscious kind of thing. This has also been ever since its introduction, the dominate factor in the development of the language. Here again it is worth noting that the idea of communication is very strongly represented. You are talking to co-designers; you are talking to yourself after a time lapse, as it were, and in general the language provides you with a tool. We have then in 1957, I believe it was, the first use of Ken's work in Harvard. In 1960 he joined IBM, and it was about a year later that I became interested and started working with the language. There were others; I don't just happen to know any other names of that period except Lyle Johnson, who was in fact our mutual manager at that time and I think one of the first people in IBM to recognize the potential of Iverson's work. The book was published in 1962. The formal description

was started in early 1963 and was completed about 15 or 18 months later with a third co-author, Ed Sussenguth. John Lawrence came along about that time, as I say, and got us interested; Peter Calingeart joined him, and all of us interacted to a considerable extent.

In 1964 we started thinking seriously about implementations. Six months later or so Larry Breed came to the company and made it all possible. He has had a leading role in every implementation, either directly or through the coding he has produced, and others. So the first implementation of APL, which you may not have heard of, was in 1965. It was a batch processor and interpretive. It ran on a 7094. It was written jointly by Larry Breed and Phil Abrams who was a consultant, and it worked. Toward the end of 1965 we were able to incorporate this under what was an experimental time sharing system, called TSM, that the Advanced Systems Development Division was working on. So, we actually had a time sharing implementation in 1965. This was all run on 7090 type machines and required special hardware, which in the Spring of '65 it was decided by whoever decides such things, that it had to be dismantled. We couldn't have it; nobody could have it; it had to go. And so we compromised. We said "Let us have a 360 instead." Through some good fortune we were in fact promised one or part of one, and in the Summer of '66 the System/360 implementation was started. By this time Dick Lathwell had joined the group and Roger Moore of I. P. Sharp was hired, and among the three of them most of the system was built in very short time. Another person who worked with them fairly closely at that time was Luther Woodrum of IBM Poughkeepsie.

I recall being out here in Endicott shortly after the formal description of System/360 was published and talking with people in the laboratory, and I see some good friendly faces around from then, like Sam Reynolds; and I guess we began to get interest in the use of APL at that time. The teaching that had been done at SRI and was continued by others, and again let me say, with considerable encouragement from John McPherson, the Director of the Institute, who always has been a friend of ours. The interest in the company was slowly growing, but it was, truthfully, not making very much headway without an implementation, and for a long time we very regretfully admitted that an implementation made a big difference after we had it. I don't feel quite so regretful any more. I like it too. So we had the 360 thing in the Summer of '66. It was completed in November, and we began to use it. The first actual application of it was, by the way, in teaching the formal methods in systems design, a course that was given at NASA (Goddard) at their invitation.

The APL\1130 was started in the Summer of '67 and it was actually the first officially released available program; it went in the type III library in the Spring of '68. It wasn't too long afterwards that the 360 version got in in August, and from there on the history is really a history of the activities of people like

you who, I presume, recognize the good parts of the language, and in the next couple days will tell us what is wrong with it, and have exploited it for your own purposes, I trust, reasonably well.

As far as the fortunes of the group itself is concerned, Ken and I weathered a large number of management changes in the department of research, which, when we joined it, was called, I think,Machine Organization; when we left was called Computer Sciences; and at the beginning of this year we moved over to the Scientific Centers which are sort of the research arm of the Data Processing Division.

As far as the present status of APL is concerned, I might say that it is alive and thriving in Data Processing.

Question: Didn't you forget in your discussion of implementations to note Herb Hellerman's contribution?

Mr. Falkoff: You are right. I mentioned Herb before, and I have a note here that I am afraid wasn't big enough. Actually, the first implementation of the notation was built by Herb Hellerman. It was the PAT system, which he subsequently published a paper on. It had, in fact, many things in it which horrified me at the time--extensions of the language which have somehow crept into what we now have and are likely to creep in still further. I think you are all very fortunate, in fact, (those of you here at Binghamton) having Herb joining your staff at this time. Ken used the PAT system in his early work with students in the local secondary school, which led to his book on elementary functions. I am sorry that I overlooked it. It is one of those things you take for granted often, things which are basic and significant.

Question: Didn't you have a typeball with a wavy underscore instead of the "∇" at one point?

Mr. Falkoff: That was our first misstep. Then we tried to change that and substitute the "∇" and change an adjacent character; I think we added the "∇" and the "Δ" at that time; and since there were two typeballs for the different kinds of terminals, the people down in Lexington built one half right and the other half right in the other direction.