

APL - Journal

A Programming Language

1-2/2016

James A. Brown

A Personal History of APL

Jon McGrew

Forgotten APL Influences

**James A. Brown and
Peter Schade**

The Evolution of Computing



IM BLICKPUNKT

Michael Baas

**Bericht von Dyalog
2016**



Liebe APL-Freunde,

ich freue mich sehr, Ihnen das APL Journal zukommen zu lassen.

In 2016 konnten wir für APL ein rundes Jubiläum begehen: 50 Jahre IBM APL. Dazu hatten sich viele APL-Freunde aus dem In- und Ausland bei der IBM in Böblingen zusammengefunden.

Eine weitere Veranstaltung hatten wir an der Hochschule in Bingen abgehalten.

Als Ergebnis haben wir erfreulicherweise auch schriftliche Beiträge für das APL Journal erhalten. An die Autoren ist deshalb ein besonderer Dank gerichtet.

Ich wünsche Ihnen viel Freude bei dem Lesen der vorliegenden Ausgabe des APL-Journals.

Ihr Dr. Reiner Nussbaum

APL Germany

INHALT

James A. Brown

A Personal History of APL

This paper contains Dr. James A. Brown's personal recollections which are accurate to the best of his ability but could be colored by 50 years of time.

Jon McGrew

Forgotten APL Influences

Each of these conferences typically focus on the future, rather than simply reviewing the past and rehash what we did years ago. I appreciate that. However, because this is the 50th anniversary of APL, my presentation is going to talk about history.

James A. Brown and Peter Schade

The Evolution of Computing

This paper is extracted from the HTML presentation made at the conference. Without the discussion that accompanied the presentation, not everything is easily understood. However, some text is added below for clarification.

Michael Baas

Bericht von Dyalog 2016

Das diesjährige Dyalog-Benutzertreffen fand vom 9-13. Oktober in Glasgow statt. Der Chronist als einer von 2 deutschen Teilnehmern möchte mit diesem Artikel gerne informieren, was da so alles geboten wurde.

Gerald Dittrich

50 years IBM APL 41 years IBM 5100

A personal computer that dominated the market from 1975 to 1983

A Personal History of APL

3

Forgotten APL Influences

21

The Evolution of Computing

55

Bericht von Dyalog 2016

64

50 years IBM APL 41 years IBM 5100

73

Bildnachweis:
Martin Barghoorn (Umschlagseite 1, 4)

James A. Brown

A Personal History of APL

On the occasion of the 50th anniversary of the APL workspace 1 CLEANSPACE German Guide Share Europe Working Group, APL Germany and IBM Germany November 27-29, 2016 IBM Böblingen, Germany¹

The Early Years Personal History

This paper contains my personal recollections which are accurate to the best of my ability but could be colored by 50 years of time. A longer version of this paper can be found on the APL Wiki at <http://aplwiki.com/OnAPLsHistory>

That version contains a discussion of various events through the years that involve people and the lighter side of technical and customer meetings including some details about the great empty array joke contest. The APLWiki paper will be updated as people involved in the early days give me corrections or additional information.

How I came to APL

I first heard of Ken Iverson, the inventor of the original APL notation, shortly after I started work in 1965 at IBM Federal Systems in Owego, New York. (My work history is summarized in Appendix 2.) A colleague (Charlie Stieglitz) was talking in the hall about this person who “reduced the design of the System 360 to a single symbol”. I thought this was a bit of an

outrageous claim (and it probably was) but I decided to check up on it. I had been interested in symbolic notations for a while and when in college read much of Alfred North Whitehead and Bertrand Russell’s “Principia Mathematica” [WI1] working out all the proofs. I went out and got Ken’s book “Automatic Data Processing” [IV1] co-authored with Fred Brooks. Then I got “A formal description of SYSTEM/360” [IV2] by Falkoff, Iverson, and Sussenguth – an amazing piece of work which later became known as the “grey manual”. I was told that when engineers were trying to fix a problem with the I/O channel, they referred to the formal description not the machine documentation. I thoroughly enjoyed time spent on this document and came to appreciate that symbolic notation really can be used for describing real as well as formal systems.

Being a newcomer in the computer world (note that the term Computer Science did

1. If you’re looking for a technical paper, this isn’t it. In this paper I document my early involvement with APL and what I learned about the early history.

not exist at this time), a friend and I decided to join the ACM (Association for Computing Machinery). One of the meetings was a dinner meeting featuring some unknown speaker and we decided to take our wives to the meeting. I'd always heard that the way to a woman's heart is through technical computer meetings. (I have no corroborating evidence that this is true.) The speaker was a man named Adin Falkoff who had a portable (by some extended meaning of that word – it came in two suitcases) 2741 terminal which he connected to a phone line and dialed into a computer at IBM Research Yorktown Heights and showed us APL.

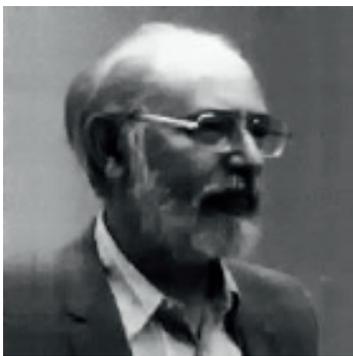


Figure: Adin Falkoff

He carefully and patiently explained to the telephone operator that the phone line would be used to communicate with a computer and the noises on the line were normal. Nevertheless, the session was interrupted several times when the connection was terminated by the operator.

I was completely mesmerized by the presentation. Adin typed in “2 space 3 backspace backspace plus” and pressed return and it typed “5”. He called it visual fidelity. I was so impressed. I recall our wives were not as impressed by this as they were apparently able to work out this result in their

heads without use of a terminal, phone line and a computer (something that would never occur to a man). A few years ago, I showed this “visual fidelity” concept again at a German APL meeting using the newly open source code of “APL\360” where I typed in “F” “backspace” “L” and displayed the value of the variable “E”. [SU1] Adin showed one wonderful thing after another and I left this meeting very excited by this brand new technology.

Imagine, a programming language without any declarations! No Declare statement – No DIMENSION statement. The number of items of data determines the size of the data structure that holds them. Unheard of. If you have zero items of data, you have an empty array. Can you imagine a DIMENSION statement for zero items?

IBM Owego had a 2741 terminal with a phone line where I could attach to the APL system at IBM Research in Yorktown Heights. Adin told us how to contact the APL operator using the system command (“) MSG”. The operator of the APL machine at Yorktown was Gerry Barrett and she added an account for me using my IBM employee number and I was off doing my first APL session. (Note that Gerry Barrett ran the APL service at IBM Yorktown and, to my knowledge was never applied to function operands to return a derived function – that’s a different kind of operator. Also note that in the fall of 1986, IBM held an “Internal Technical Liaison” ITL meeting for the 20th anniversary of APL and Gerry was presented with an award.) After I had been playing with APL for a while, Gerry sent me another message asking for an account number to

which my usage could be charged. “Oh Oh” I thought and quickly signed “)OFF”. I asked my manager for an account number and he was completely uninterested in letting me “play on company time.”

That might have been the end of my time with APL but one of the benefits that IBM offered as part of employment was the ability to take graduate courses at Syracuse University for free. They would fly professors from Syracuse to Endicott, New York and we could take classes one evening a week. One of the teachers, Bill Jones, mentioned how Syracuse University was going to get an APL installation. (I’m pretty sure the only other APL installation outside of Yorktown at this time was at a college in Canada.)

I decided I would go back to school at Syracuse and work on APL. IBM provided grants for employees to get advanced degrees so I applied for a grant. I was denied so I took a leave of absence from my job at IBM Owego to go on campus full time and run their APL service for a salary of \$500 per month. (Students give Universities a really cheap work force.) It is, perhaps, interesting that my reading of “Principia Mathematica” [WI1] was accepted as fulfilling the mathematical logic requirement for my doctorate. Syracuse also had a foreign language requirement for a PhD and I always thought I would use German but it turns out that, at that time, FORTRAN was one of the accepted languages. I guess it’s foreign enough. (By the way, more FORTRAN programmers have died than APL programmers. That has to tell you something!)

I was at Syracuse when Al Rose visited with his famous “portable” 2741 (120

pounds - maybe the same one Adin used the first time I saw him). Sometimes when Al wanted to give a demo of APL, he would be unable to get a phone connection to Yorktown Heights. He solved this problem by noting that the electric typewriter used an acoustic coupler to communicate with the computer. He hooked up a cassette tape recorder to the coupler and recorded the tones from the coupler while he was giving a live demo. Now when he couldn’t get a phone line, he played the tape into the coupler and it reproduced the demo session he had recorded in real time – errors and everything. This is memorialized in the song “APL Blossom Time” written by Mike Montalbano [MO3].

Fortunately (in some perverse meaning of that word), Karen and I ran out of money after six months at Syracuse and, in the spring of 1969, I had to get a second job. I applied to IBM Yorktown Research for a summer job. (Note that this application was done by writing words on paper, putting the paper in an envelope with a stamp and sending it though the Post Office. This was called a letter. There’s a whole generation of people now who could not imagine such an archaic way to communicate.) Despite this cumbersome way to apply for a job, my application was quickly rejected. Again, fortunately, I had also sent a letter directly to Adin Falkoff of the APL group and they invited me for an interview.

Karen and I showed up at Yorktown Heights on the appointed day and time only to find that the interview had been canceled. Because of the death of President Eisenhower, the lab had been closed the

previous day and a scheduled “APL Machine” seminar had been moved one day later. They had tried to reach me to cancel the interview but did not succeed. (Why didn’t they send an Email or call me on my cell phone?) But I did see Ken in the flesh for the first time standing outside the auditorium.



Figure: Ken Iverson

I wish I could have attended the APL Machine Seminar. I’m told that central to the discussion was Phil Abram’s PhD dissertation “An APL Machine” [AB1]. His concepts of drag along and beating made it into the implementation of APL2. It’s always felt like a hardware implementation of APL was a natural idea. Even today, NestedComputing and the “Array Operating System” first suggested by Mircea Morosan are being pursued and might influence hardware.

Karen and I came back to the Yorktown lab the next day for the interview and I met Ken Iverson, Adin Falkoff, Larry Breed, Dick Lathwell, and the team for the first time. I got the job and spent the summer at the lab with the APL team. I know now that Ken and Adin sort of took turns managing the group but the focus was always on the work not the management. I’ve told this story many times before but I had assumed that Adin was my manager. It turns out that

Larry Breed was actually my manager but I didn’t find out until the exit interview at the end of the summer. (As a strange twist, many years later Larry worked for me at IBM in Palo Alto. I hope he realized it.) Because Larry made an administrative error, my summer job was not terminated and each summer I would go back to Yorktown for a few months and I continued for three years working remotely from the local IBM office in Syracuse during the school year. In those years, among other things, I implemented the extension of “base value” and “representation” (now called “decode” and “encode”) to higher rank array arguments and “catenate” on higher rank arrays (it only worked on vectors before).

There was a problem with the definition of “base value” and “representation”. Because “representation” only worked on scalars, the implementation treated a one item vector right argument as a scalar:

```

                2 2 2 T 5
1 0 1
                2 2 2 T, 5
1 0 1
    
```

When extending to higher rank arrays, with the desired shape equation, the result of “representation” of a one element vector would change to be a one-column matrix:

```

                2 2 2 T 5
1 0 1
                2 2 2 T, 5
1
0
1
    
```

We were running a time sharing service and having a primitive function suddenly get a different answer was a potential problem. To measure the impact of the change, we put counters in the code to count the number of times “representation” was applied to a one-item vector and the number of times it was applied to a scalar. At the end of the day, the counter showed there were many times more of application of “representation” to one-item vectors than to scalars. Way more than expected and a big problem. We speculated that this could be because the shape of a vector was itself a one-item vector. Maybe one-item vectors were more common than we thought. Then, I discovered that another summer employee, Seth Breidbart, overheard us talking about the counters and put two terminals into infinite loops taking the “representation” of one-item vectors. This was going to be my first assignment and I really wanted to do it so I fired up three terminals taking representation of scalars and after a while, my counter surpassed his counter by enough that I was given the go-ahead to do the implementation.

Not many people had the ability to work on computers remotely in those days. I was using a time sharing system called “Time Sharing System” (TSS) and it turns out that for months, my usage was not billed to the APL group. There was a bug in “free-main” the API that released allocated storage. Each time I logged off, TSS crashed as (I assume) my workspace was released. I was logging off so I didn’t notice anything unusual. Because session accounting was done on log off, the APL group were never billed for my usage. Eventually the TSS

system programmers noticed that in every core dump, my ID was on the system.

In the original APL, an array was either all numeric or all character. I met Trenchard More during a summer at Yorktown and heard discussions of extending APL. I left IBM that summer with the idea that there was no need for arrays to be all numeric or all character. Since I decided to work on extended APL for my PhD Dissertation, I was (quite properly) not permitted access to the proposals that were active at Yorktown in particular the work of Trenchard. My dissertation “A Generalization of APL” [BR1] was published in 1971 and contained arrays that could contain some numbers some characters and recursively some arrays. I graduated from Syracuse in the first computer science class with a PhD in Computer Science and moved to IBM at the Philadelphia Scientific Center where the first lines of code for APL2 were written.

Stories from the early years

Much of what I’ll say in these sections is not firsthand knowledge but rather things told to me or overheard by me. Someone with more personal knowledge should corroborate.

For some time in the early years, the term “Iverson Notation” was used to describe the symbolic notation invented by Ken Iverson. At some point, the team decided that they needed to give a formal name to the notation. I was told that many names were proposed and most people hated most of them. I wish someone had recorded the names

that were suggested but only the final name survives. I was told that it was Adin who suggested taking the initials of Ken's book "A Programming Language" [IV3] and APL was born. To this day we still need to point out that the word "Programming" in the title of Ken's book did not refer to "Computer Programming" but to "Mathematical Programming" and "Linear Programming". I wonder how the computing world would be different if they had chosen the name "Java".

The first implementation of Iverson notation was done by Herb Hellerman in a system called "PAT" (Personalized Array Translator System) [HE1] – an interactive (but not time-shared) system on an IBM 1620 not using Greek characters. (I have met Herb Hellerman and I can testify that this picture does not do him justice but it's the best I could find.)



Figure: Herb Hellerman

Ken used Hellerman's implementation with students in the local secondary school. This led to his book on elementary functions [IV4].

Not long after this, the IBM Selectric typewriter became available and a type element with the symbols used in Iverson Notation was created.



Figure: IBM Selectric typewriter

Creation of the type ball was complicated because opposite sides of the ball had to have characters of the same density so that balance was preserved. I am told that Herb Hellerman called these "Iverson Balls" and this term was used until Ken himself requested that they be called "Iverson Type Elements".



Figure: "Iverson Type Elements"

When I first arrived at Syracuse University to start graduate school in earnest, there was not yet a local installation of APL. They had a 2741 and a phone with an acoustic coupler just like they did at IBM Owego. There was a problem with using this terminal because the type ball was often missing. You removed the type ball just by lifting the clip on the top. It became popular for the young female students to steal the type ball and wear it on a necklace around their neck as a love bead. (Remember this was the 1960's.). The solution was to break off the clip on the type ball making it difficult

to remove. Garth Foster (who became my Thesis advisor) was the holder of the type ball and you had to reserve time on the terminal.

In 1965, Larry Breed and Phil Abrams built a version of APL in FORTRAN on an IBM 7093 at Stanford University. This was around the time that the “Formal Description of System 360” became available. I had read this paper and I didn’t find any errors. But Larry told me that he did find an error in the formal description and when Ken visited Stanford (maybe to see their APL implementation – this I don’t know for sure) he and Phil had lunch with Ken. Larry said he waited until Ken had cake in his mouth and then told him about the error. It was probably this meeting that led Larry to come to IBM Yorktown to work on APL.

The APL implementation that I used from IBM Owego and Syracuse was hosted at IBM Yorktown on an IBM 360 model 50. This machine had a total of 256K of memory. It ran on the DOS operating system (not the DOS from PC days) and when APL was loaded it replaced part of the operating system because there was not room for both APL and the DOS operating system. Users were given 30K workspaces which were swapped from disk to memory and back as users computed. APL is such a compact notation that you can do significant work in a 30K workspace. No one had ever seen anything like it.

When you wanted to attach to the APL service, you called a toll free number and, when you heard a whistling sound, you put the phone into an Acoustic Coupler.



Figure: Phone with Acoustic Coupler

The back of the computer room in Yorktown was a wall of telephones to receive these calls. When the APL service was unavailable, dozens of phones would be incessantly ringing. I was in the machine room with Graham Driscoll (a member of the APL group) during one of these down times. He would go back to the phones and pick them up and make a whistling sound with his mouth and I suspect most people put the phone on their side into the coupler thinking the computer had answered and tried to sign on.

The APL service at Yorktown ran 24 hours a day – something that was not only not normal at that time but at Yorktown not even allowed. At night, the computer ran unattended and that was forbidden. The APL group did it anyway and was continually bothered by management because of it.

At some point, management decided that the machine should be shared with another group in the lab. It was, after all, a valuable resource. I’m told that Adin told Eugene McDonnell to go to the machine room and guard the door. When people from the other group showed up to use their

machine time, Eugene physically prevented them from entering the room. This led to a big problem and eventually, the APL team lost the fight for the machine. I'm told that because of this, Ken Iverson resigned from IBM. The team had a going away party for Ken and at the party, someone from IBM management talked to Ken and offered to let him keep the machine. (This might be when Ken was made an IBM Fellow but I am not certain about that.) The APL group had its own machine for the rest of its time at Yorktown and when the group moved to form the Philadelphia Scientific Center, they again had their own computer.

IBM Thomas J. Watson Research Center Yorktown Heights

The building is in the shape of an arc of a circle on a rising hill and has dark glass on the front. Office corridors run from the front to the back so no office has a window - a big advantage if you want to grow mushrooms. There are three floors but the city of Yorktown only allows buildings with two floors. Because of this, the official main entrance is in the back of the building higher on the hill so from the back it is a two floor building. The bottom floor was floor "0". In the USA, the ground floor of a building is normally floor 1. Even at Yorktown, zero-origin indexing was preferred. It makes sense to me that floors are numbered by their distance from the ground not that distance plus one. Here's a picture of the real



main entrance to the lab in the "back" (? "front") of the building (see photo below).

It was a joy to work at Yorktown heights. The work was important and little else was. At IBM Research there was only one job for technical people "Staff member". Everyone was treated equally. That's probably why I didn't know who my manager was. Every year all the staff members were ranked from first to last and if you ranked in the lower third two years in a row, you had to find another job.

The head of research at that time was a Mathematician (name forgotten) and the one "unequal" thing he insisted was that the mathematicians at the lab had larger chalk boards than the other employees. The APL group held that this was needed because of their cumbersome notation. It is often reported, but not true, that the APL group had small thin chalk boards suitable for writing APL one liners.

There's one memo from management that I wish I had saved. The APL group played Frisbee at lunch time and the black 150

*Figure: IBM Thomas J. Watson Research Center
Yorktown Heights*

gram Frisbee was registered with the IFC. If it was raining, we sometimes played Frisbee in the hallway. This left some black streaks on the wall on those rare occasions we did not execute a perfect toss. Management sent a memo saying “Only white Frisbees indoors”. How cool.

Library 1 CLEANSPLACE

The first version of APL had no library system. You signed on, typed in your program, played with it and all was lost when you signed off. If, in the middle of a session, you wanted to start over, you would sign off and back on again and again have an empty workspace. Since there were a limited number of phone lines, you could not always get back on immediately. When System commands and a library system were added in 1966, the first workspace saved was library 1 CLEANSPLACE with the now famous timestamp of 1966-11-27 17.53.58. This workspace had no functions or variables and so was formatted as a workspace but otherwise completely empty. Now you could save your work in your own private library. There was no)CLEAR command. If you wanted to start over with an empty workspace, you loaded 1 CLEANSPLACE. Much better than signing off and back on.

When)CLEAR, to give you an empty workspace, was added, CLEANSPLACE was no longer needed so Adin deleted it by entering “)DROP 1 CLEANSPLACE”. He had second thoughts and decided CLEANSPLACE should be kept for historical reasons and asked Dick Lathwell to add it back. Dick set the clock of the model 50 to the original timestamp with the clock disabled

and started APL and entered “)SAVE 1 CLEANSPLACE” However, an interval timer interrupt was required to enter the scheduler, so he momentarily enabled the clock. The SAVE happened but the clock advanced one second so the timestamp we see now is one second later than the original. [JS1]

When APL does operations on numbers (like comparisons or conversions) it uses fuzzy operations so, for example, the number 1.9999999999999999 is treated as 2 if it is used as an index. The implementation had a constant called Comparison Tolerance (CT) stored in the bottom of the workspace and set to 1E-13 that was used in these adjustments. There was no way for a user to adjust this constant. However, for some scientific calculations, you do not want any adjustment to numbers so a workspace 1 HEISENBERG was saved with Comparison Tolerance set to zero so there would be no uncertainty in computations. This workspace never received the acclaim of CLEANSPLACE. In modern APL systems, users can set any reasonable value of Comparison Tolerance. HEISENBERG was no longer needed and was dropped. It was never recreated. Where’s Dick Lathwell when you need him?

Vector Notation

I have a printout from an early APL session (from before I joined the group) and it shows expressions like this:

$$(2, 3, 4, 5) + (20, 30, 40, 50)$$

$$(22, 33, 44, 55)$$

Notice how numeric vectors were written like you might see them in mathematical textbooks you would see at that time. Larry Breed was very proud of the code he wrote in the parser which could recognize the parenthesized expression as a numeric constant and store it internally in the APL workspace as a single token with a vector value. APL programs are executed by an interpreter that at execution time must examine every token so Larry's enhancement meant faster execution because it could scan one token instead of many.

I was told that it was a very painful decision to change the syntax of APL so that the parentheses and commas were not needed.

2 3 4 5+20 30 40 50

22 33 44 55

After all, unless you know the rules are otherwise, it sure looks like there is the expression "5+20" in the middle. The decision was made that forming of vectors was more important than the application of operations.

In my opinion, this was a revolutionary step in the development of the language. There was already the idea that the less dense characters (those that used the least ink) should be for the most important uses. Using '?' for an operator uses this principle. What's the least dense character? It's the space character. And now it is being used for the most important role - forming vectors. It's interesting that the least dense character is assigned to the largest key on the keyboard!

When I was designing the APL2 language, without going into details, I decided again that forming of vectors was more important than the application of operations so if A, B, C, D, E, F are any arbitrary arrays,

$$A \ B \ C+D \ E \ F$$

means

$$(A+D) (B+E) (C+F) .$$

This decision (prompted by the work of Trenchard More in his Array Theory [MO2]) led to the most difficult arguments between the originators of APL and me. I always looked for a compromise that would make all parties happy but in this case, it was not possible.

When I moved from APL Development at IBM Santa Teresa in California back to IBM Research in Yorktown Heights, the California group was worried that I'd give up on vector notation. Gene McDonnell said to me before I left "We will know you have caved in when Vector Notation disappears". I didn't cave (painful though that was) and when Gene moved to I.P. Sharpe, he changed his mind about vector notation.

One objection to this vector notation expressed during the decision making process is that more expressions become unreadable without redundant parentheses. I believe this argument has some validity but, for me, other arguments carried the day. This is why, once formal requirements

are satisfied, language design is an art not a science.

People have said that APL programs are unreadable. Some have even called APL a write only language. Alan Perlis (famous American computer scientist who was on the ALGOL team) once said this about APL and the APL interpreter:

“APL is like the Bible. It is not meant merely to be read but to be interpreted.”

It was a clever play on words.

In the early days, when the universe of APL people was very small, decisions were made by consensus. I remember taking ideas to Adin. He would pick them apart and tell me how terrible they were (this was Adin’s style). I’d take the same ideas to Ken. He would pick them apart and tell be how promising they were (this was Ken’s style). As days went on and ideas were discussed, slowly Adin’s and Ken’s opinions would converge and almost every time their merged opinions were clearly correct. (This balance was lost when Ken left the group years later.) As APL became popular and more people became involved it became more difficult, and eventually impossible, to reach consensus on decisions that had to be made.

The differences of opinion over my “Vector notation”, operator definitions and nesting were never resolved and led to two styles of array computing from different companies. You’re not right because people agree with you and you’re not wrong because people disagree with you. You have to make technical decisions without emotion

based on formal arguments. You should ask for advice from a lot of people but not so they can tell you the answer. Rather you ask so that you have the alternatives you need to make a decision.

Another case where I made a compromise was in the 1982 APL2 IUP where, in an attempt to gain agreement, I changed the definition of some operators (notably Outer Product) to a more flat orientation. This seemed reasonable because the older behavior I wanted could be achieved my applying “each” to the operand. This was a terrible decision.

In 1982, IBM and STSC announced their enhanced APL products on the same day. Here’s an article from ComputerWorld:



Figure: Article about enhanced APL products

STSC’s had the definitions of the operators from my PhD thesis and IBM’s did not. Thanks to Bob Smith, who visited IBM Santa Teresa sometime after he left STSC, the decision was reversed before the release of the official Program Product two years later. Bob Smith is also responsible for some of the nomenclature. I called the extended arrays “General Arrays”. Bob called them “Nested Arrays” and I felt this name was more descriptive and adopted it.

I'm very proud of the final definition of the APL2 language and its correctness is due to a myriad of interested and talented people.

You can find a chronology of APL Systems and influences at <http://www.sigapl.org/APLChronology.php>.

APL Blossom Time

The song "APL Blossom Time" was written by Mike Montalbano using the pseudonym J.C.L Guest. Mike used a pen name for many of his writings. He wrote several articles for "Datamation", a computer magazine that was published in print form in the United States between 1957 and 1998 and still continues on the web [Wikipedia]. These articles were viewed as criticism of IBM management and I was told by Mike that for many years, IBM was trying to find the person who wrote those articles. He probably didn't need to use his fake name for "APL Blossom Time" but he did.

The lyrics are a very accurate representation of the early years of APL. A discussion of the writing of the song is included in Mike's "A Personal History of APL" [MO1]. I discussed earlier in this paper seeing Al Rose "fed it a tape when he couldn't get a phone line" at Syracuse.

At APL 81 in San Francisco, Larry Breed, John Bunda, Diana Dlouhy, Al O'Hara, Rob Skinner and I performed this song at the banquet with 1100 people singing along. Someone unknown to me took this picture of us performing the song (see photo). A YouTube video that contains the lyrics and

the audio of this performance is viewable at [MO4]:

<https://youtu.be/g4xvjfr297E>



Figure: Larry Breed, John Bunda, Diana Dlouhy, Al O'Hara, Rob Skinner and James A. Brown

The quality of the recording is not bad considering that the recording was made by Bob Armstrong sitting in the audience using a hand held cassette recorder. We had the lyrics on foils (transparencies that could be projected by an overhead projector) and, in the introduction to the song, I say how we had a musical interlude between verses so the foils could be swapped. I still have the original foils.

Sometime after APL 81, I made a recording of the song on a multitrack recorder at my home with vocals by me and Mike Wheatley, guitar by me and John Bunda, Bass by me, and drums by Brian Duff. A YouTube video that contains some historical pictures and this audio can be found at [MO3]:

<https://youtu.be/0zSauxkMxPo>

As of November 2016 this video already has over 60 views and is on its way to becoming a viral video. There is (at least) one glaring error in the video. See if you can spot it. I would like to claim that it is not an error but rather copyright protection. At one point in APL history, a company stole the source code for VS APL and claimed it as their own. Doug Aiton, a member of the APL team, was charged with showing that the code was stolen. This was done by cataloging the bugs in the product over the years. By identifying which bugs were fixed and which were not, Doug was able to pinpoint the date on which the code was stolen within a couple of weeks. Should someone steal my video of APL Blossom Time, I can prove it by pointing out the errors. However, it's not clear what intellectual property is being protected. Mike Montalbono gave us permission to use his words. The music was written in 1936 by Jimmy Driftwood for a song called "The Battle of New Orleans". I assume the music in the public domain even though there was a popular rendition of the song by Johnny Horton which went to number 1 on the Billboard Hot 100 in 1959 [Wikipedia].

The Future for me

After 31 years, I retired from IBM (as of this writing 20 years ago) in 1996 because of a headcount cut that would have caused others in the team to be let go if I didn't leave. I've continued to work to move the array programming paradigm represented by APL into mainstream computing. Together with James Wheeler, SmartArrays Inc. produced an add-in to C++, C#, and JAVA that gives these languages array computing

capabilities similar to and beyond what APL provides. With Mircea Morosan, Morten Kromberg and Gitte Christensen, NestedComputing has a possibility of moving array computing deeper into operating systems and even into the hardware. Array representation of data and array processing may play a part in the emergence of a new generation of computers that have flat address spaces.

Conclusion

Warren Buffett once said "Always associate yourself with people who are better than you." I certainly did this when I began to associate with APL in 1967 and join the group in 1969 and I've continued to do this whenever the opportunity was available. It was and is an honor and a pleasure to work with APL and the talented people who are drawn to APL.

Acknowledgements

So many people contributed to the design, implementation, support and marketing of APL and APL2 that it is impossible to give even a partial list. Nevertheless, here's a few citations with apologies to those not mentioned.

Of course Ken Iverson and Adin Falkoff must be mentioned. Even though Adin and I had serious disagreements on the language definition, he was always a good manager and furthered my career. Many people from the early APL group need to be mentioned. Larry Breed showed me that the work is important and, as I said, I did not know he was my manager. Eugene

McDonnell showed me how to use mathematics in the design of notation. Trenchard More never lost his cool and presented arguments with equations not emotion. He had a tremendous influence on the final definition of the APL2 language. His array theory became the theoretical basis for the arrays in APL2. Dick Lathwell showed me a certain indifference to the chain of command. When I left Syracuse and came back to IBM at the IBM Philadelphia Scientific Center, Dick was my manager. He was a great manager and I believe we did significant and valuable work together, He was also fun to be around. He advised me to take positions strongly and then, if nobody objects, you've won. If someone takes a strong counter-position and it has some merit, back off. (This absolutely the right thing to do. You don't think so? Nevermind.) Dick was also very influential in my personal life.

Garth Foster, my thesis advisor at Syracuse University, was a guiding influence throughout the creation of the first version of what became APL2 as represented in my thesis "A Generalization of APL" [BR1]. I avoided many potential trouble areas because of his close attention. It was very pleasing to me that Garth used my dissertation as study material for some classes for several years after I graduated. Garth has continued to host APL implementer's conferences at the Syracuse Minnowbrook Conference Center.

As mentioned before, Bob Smith, when he visited the Santa Teresa Lab, forced me to face the fact that simplicity and elegance are more important than consensus. He continues to do exciting work with his

Nested Arrays Research System NARS2000 (<http://www.nars2000.org/>) which is in the public domain.

My wife, Karen, (best friend for over 55 years) has been through it all with me. She's been to so many of my presentations that she could probably give some of them. At APL81, in Washington, D.C., she overheard someone pointing her out by saying "There's the other Mrs. Nested Arrays". We believe this was Bob Smith's wife, Mary, who was called "Mrs. Nested Arrays" by his STSC colleagues.

Finally, I must thank Axel Güth for being my mentor for technical, business and marketing matters. This began when we first met and continues to this day. We still have long phone calls to discuss technical and business matters. He remains my only colleague who has a separate mail folder in my Email client:

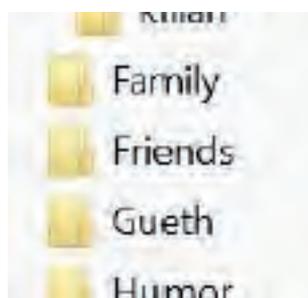


Figure: The author's mail folder

There is no significance that his folder comes before the "Humor" folder. It is an accident of alphabetical order.

References

[AB1] Abrams, P; "An APL Machine"; PhD

- Dissertation Stanford University; 1970
- [BR1] Brown, J.; "A Generalization of APL"; PhD Dissertation Syracuse University School of Computer Science; 1971
- [HE1] Hellerman, H.; "Experimental personalized array translator system"; Communications of the ACM; Volume 7 Issue 7, July 1964; Pages 433-438
- [IV1] Iverson, K., & Brooks, P.; "Automatic Data Processing"; John Wiley and Sons; 1963
- [IV2] Iverson, K., Falkoff, A., Sussenguth E.; "A Formal Description of SYSTEM/360"; IBM Systems Journal; Vol. 3 No. 3, 1964
- [IV3] Iverson, K.; "A Programming Language"; John Wiley & Sons, Inc.; 1964
- [IV4] Iverson, K." Elementary Functions: An Algorithmic Treatment"; Science Research Associates; 1966
- [JS1] Various; "APL Quotations and Anecdotes"; <http://www.jssoftware.com/papers/APLQA.htm>
- [MO1] Montalbano, M.; "A Personal History of APL"; <http://ed-thelen.org/comp-hist/APL-hist.html>
- [MO2] More, T; "Notes on the Development of a theory of Arrays"; IBM Philadelphia Scientific Center Tech, Report No. 320-3016, May 1973
- [MO3] Montalbano, M. (Guest, J.C.L); "APL Blossom Time"; YouTube video <https://youtu.be/0zSauxkMxPo>
- [MO4] Montalbano, M. (Guest, J.C.L); "APL Blossom Time Lyrics"; YouTube video <https://youtu.be/g4xvjfr297E>
- [PO1] Post, Ed.; "Real Programmers Don't Use Pascal"; Letter to the editor, Datamation July 1983
- [SU1] Sushtek, L, "The APL Programming Language Source Code", <http://www.computerhistory.org/atcm/the-apl-programming-language-source-code/>
- [WI1] Whitehead, A.N., Russell, B; "Principia Mathematica"; Cambridge at the University Press, 1910.

Appendix 1: APL Blossom Time Lyrics

APL Blossom Time

J. C. L. Guest (Michael S. Montalbano)

Back in the old days, in 1962,
A feller named Ken Iverson decided what
to do.
He gathered all the papers he'd been writing
for a spell
And he put them in a little book and called
it APL.

Well...

He got him a jot and he got him a ravel
And he revved his compression up as high
as she could go
And he did some reduction and he did
some expansion
And he sheltered all his numbers with a
ceiling and a flo'

Now Sussenguth and Falkoff, they thought
it would be fine
To use the new notation to describe the
product line.
They got with Dr. Iverson and went behind
the scenes
And wrote a clear description of a batch of
new machines.

Well...

They wrote down dots and they wrote
down squiggles
And they wrote down symbols that they
didn't even know

And they wrote down questions when they
didn't know the answer
And they made the Systems Journal in
nineteen sixty-fo'

Now writing dots and squiggles is a mighty
pleasant task
But it doesn't answer questions that a lot of
people ask.
Ken needed an interpreter for folks who
couldn't read
So he hiked to Californ-i-a to talk to Larry
Breed.

Oh, he got Larry Breed and he got Phil
Abrams
And they started coding Fortran just as fast
as they could go
And they punched up cards and ran them
through the reader
In Stanford, Palo Alto, on the seventy
ninety oh.

Well a Fortran batch interpreter's a mighty
awesome thing
But while it hums a pretty tune it doesn't
really sing.
The thing that we all had to have to make
our lives sublime
Was an interactive program that would
let us share the time.

Oh, they got Roger Moore and they got
Dick Lathwell,
And they got Gene McDonnell with his
carets and his sticks,
And you should've heard the uproar in the
Hudson River valley
When they saved the first CLEANSPACE
in 1966.

Well, when Al Rose saw this he took a little ride
In a big station wagon with a type ball by
his side.
He did a lot of teaching and he had a lot of fun
With an old, bent, beat-up 2741.

Oh, it typed out stars and it typed out
circles
And it twisted and it wiggled just like a
living thing.
Al fed it a tape when he couldn't get a
phone line
And it purred like a tiger with its trainer
in the ring.

Now, there's much more to the story, but I
just don't have the time
(And I doubt you have the patience) for an
even longer rhyme. So I'm ending this first
chapter of the tale I hope to tell
Of how Iverson's notation blossomed into
APL.

So..
Keep writing nands when you're not writ-
ing neithers,
And point with an arrow to the place you
want to be,
But don't forget to bless those early APL
sources
Who preserved the little seedling that be-
came an APL tree.

Dedicated to the pioneers of APL with re-
spect and affection by J. C. L. Guest

Appendix 2: Work History

1965-1968 IBM Owego, Owego New York

1968-1971 Syracuse University, Syracuse, New York
 1969-1971 IBM Research Yorktown Heights, New York
 1971-1974 IBM Philadelphia Scientific Center, Philadelphia, PA
 1974-1974 University of Pennsylvania Moore School of Engineering, Philadelphia, PA Professor
 1974-1978 IBM Palo Alto, Palo Alto, Ca.
 1978-1981 IBM Research Yorktown Heights, New York
 1981-1997 IBM Palo Alto and IBM Santa Teresa, San Jose, California
 1999- present SmartArrays Inc, Ladera Ranch, California
 2004-2007 USAA, San Antonio, Texas
 2010- present NestedComputing Corp, Ladera Ranch, California

Appendix 3: Cast of Characters

These are people mentioned in the “Early Years” portion of this paper with a few words about what they did and what influence they had on me. Apologies to the innumerable people who influenced my work who did not get a mention.

▶ **Ken Iverson**

Creator of the original APL and was the moral leader of the APL community for the rest of his life.

▶ **Adin Falkoff**

Adin’s talk at an ACM meeting was my first encounter with APL. He was my manager at IBM several times over the years as I moved back and forth from the east to the west to the east to the west coast of the US.

▶ **Edward Sussenguth**

Collaborator on “The Formal Description of System 360“. To my knowledge, he had no further influence on the APL world. I never met him.

▶ **Gerry Barrett**

Operated the IBM 360 Model 50 on which the first APL service ran.

▶ **Bill Jones**

Professor at Syracuse University who taught at IBM Endicott. Responsible for me going to Syracuse.

▶ **Al Rose**

First marketer of APL. Famous for his portable 2741 and the ability to give an APL demo even when he could not attach to the APL service.

▶ **Mike Montalbano**

Proponent of APL from the early days. Author of the lyrics of the song “APL Blossom Time“. Member of the APL Development group in Northern California.

▶ **Phil Abrams**

Wrote an early APL system with Larry Breed at Stanford University. His PhD Thesis “An APL Machine” contained execution schemes which I incorporated in the implementation of APL2.

▶ **Larry Breed**

My first APL manager (unbeknownst to me) and implementer of what became APL360. Moved to STSC (Scientific Time Sharing) to work on their service. Later came back to IBM and worked for me in Palo Alto.

▶ **Dick Lathwell**

Implementer of many important aspects of IBM APL (including shared Variables). Was my manager at the IBM Philadelphia Scientific Center and became a close personal friend.

▶ **Seth Breidbart**

Another summer employee at IBM Research in Yorktown heights. He was known for a dry sense of humor. If you said “The sun is hot today” he’d say “Yes. Several million degrees”. He has continued to produce interesting work both in the APL world and other disciplines.

▶ **Trenchard More**

Creator of “Array Theory”. His theory and his style of work had a tremendous influence on APL2 and on me personally. He is a true computer scientist who applies rigor to everything he does.

▶ **Herb Hellerman**

Author of the first “Iverson Notation” computer implementation PAT (Personalized Array Translator System).

▶ **Garth Foster**

My PhD thesis advisor at Syracuse University. Instrumental in the spread of APL and creator of the APL Quote Quad - the APL periodical. He still runs APL Technical meetings at the Syracuse University Minnowbrook Conference Center.

▶ **Graham Driscoll**

Member of the early APL group at IBM Yorktown.

▶ **Eugene McDonnell**

Member of the early APL group at IBM Yorktown. Gene was responsible for many of the numerical primitives in the APL language notable the circle functions and computing on complex numbers.

▶ **Alan Perlis**

A computer scientist who was the first person to receive the Turing Award. He was on the team that developed ALGOL. As a professor at Yale University, he was interested in APL and wrote many papers including “Should APL and Lisp be combined”. The answer was NO!



Contact:

Dr. James A.
Brown,
NESTEDCOM-
PUTING COR-
PORATION,
28 Drackert Ln,
Ladera Ranch,
CA 92694

Jon McGrew

Forgotten APL Influences

Each of these conferences typically focus on the future, rather than simply reviewing the past and rehash what we did years ago. I appreciate that. However, because this is the 50th anniversary of APL, my presentation is going to talk about history.

I realized that there are a lot of things from the past for which APL really should get high marks, some of which seem to be forgotten. These are places where APL has really made its mark and made an influence in the world around us, but it may have been forgotten that APL was ever involved with that. We all use instant messaging, word processors, and spreadsheets... but are you aware that these all have links to APL?¹

Before I get into that, I have a list of *Thank You's*, and one of the things that I will start with is workspace 1 *CLEANSPACE*. I put out an APL newsletter in the 1970s and '80s, internal to IBM, called *The APL Jot Dot Times*, and in the mid-1970s, I wrote an article for it about the history of APL at that time, and one question that I wanted to address was, “*When was APL ‘born’?*”

That turned out to be a more complicated question than I had expected, so finding a good answer became kind of a quest for a while. The problem was, should we consider its starting point to be when Ken Iverson first started to come up with ideas for his notation, *or* when he started using the material for teaching at Harvard, *or* when he sent his seminal book to press (“*A Programming Language*”^[1,2,3]), *or* when the publisher first made that book available to the public, *or* when he joined IBM, *or* when his group first started implementing it on the computer, *or* when the first product was released, ...*or*...?

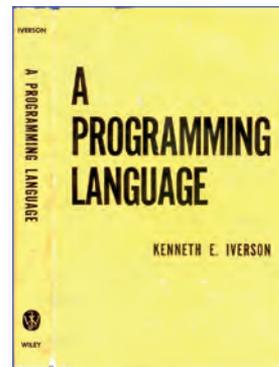


Figure: “*A Programming Language*” textbook, 1962

There are so many dates that we could have chosen for this—all of them reasonable. And finally, I realized that the first workspace that the developers saved was *still there*, and for the users of our time-sharing systems which we were running there in Kingston, New York—the people who would be receiving my newsletter— I thought that the more important point *to them* is the history of *what they are actually using*, not necessarily the notation that preceded that, so back in the mid-1970s I chose workspace 1 *CLEANSPACE* as being a starting point for APL:

¹ Given as a multimedia presentation at the APL 50th Anniversary Conference at Böblingen, Germany on 29 November 2016

```
)LOAD 1 CLEANSPACE  
SAVED 1966-11-27 17.53.59 (GMT-5)
```

I have noticed that some people have also pointed to the 1991 issue of the *IBM Systems Journal*^[4] which celebrated the 25th Anniversary of APL, saying that this underscores the validity of using 1966 as the starting point for APL. I agree; thank you—that was the issue which Ray Polivka and I created for that event.

Michael S. Montalbano also had some discussion of workspace 1 *CLEANSPACE* in his 1982 paper, “*A Personal History of APL*.”^[5]

I’m not sure how many other people also started looking at APL’s “date of birth” back in the 1970s. But I was pleased to see that this conference observed that as the starting point for APL.



So I am going to start with some background by just talking about some of the people involved, with a huge *Thank You* to each of them:



Figure: Original APL\360 Design Group

This photo^[6] shows members of the original APL\360 Design Group and probably most of you know the players, but for those of you who don’t, Ken Iverson and Adin Falkoff, of course, were the key players who were in charge of designing and creating the APL language, and those who actually did the implementation work were Dick Lathwell, Roger Moore, Phil Abrams, and Larry Breed. They were the original development team for APL\360. This is a picture that was taken at the I. P. Sharp Conference in Toronto in 1982.

...And by the way, just as an aside, does it ever bother you how people sometimes get rather arbitrarily cropped out of pictures?



Figure: Photobombed?! ... “Which one is not like the others? Which one doesn’t belong?”

A friend showed me this uncropped photo,^[7] and... well, okay, so it wasn’t actually all *that* arbitrary in this case; I’ll have to admit that that’s me skulking about in the background, lurking there in the shadows— but I was never part of the APL\360 Design Group. This photo was taken in the conference hospitality suite, and I just happened to be coming in for coffee. I had no idea they were taking a picture there that day, and I certainly had no intention of photobombing anyone. Oops.

Another piece of history that I'll show you is this photo^[8] from an ACM meeting in Las Vegas in August of 1968. IBM put quite a large booth together, and notice that there are several IBM 2741 terminals^[9,10] set up, each with television cameras pointed at them and monitors above them so that people could see what was being typed on the page; this was back before video display terminals were as commonly available:



Front: Arnold, Sandra Pakin, Conroy, Adin Falkoff; **Back:** Crane, B. Bergquist, T. Wilson, Van Guilderan, Al Rose, Mike Montalbano

Figure: APL at ACM meeting in Las Vegas in 1968

None of this so far has anything to do with the *Forgotten APL Influences*, but because this is a 50th Anniversary Conference, I thought that we ought to observe some of our collective APL backgrounds.

~

■ My background

Here's a little bit about my own background:

- I am retired from IBM.
- I have been using APL since 1971.
- APL became my career in 1975.
- I worked in the APL support team for a large timesharing system in Kingston, New York. We had users in 17 countries using our machine.
- I then worked on APL projects in the group called *Numerically Intensive Computing*. More about that in a bit.
- I was a member of the APL Development team at IBM during the development of APL2.^[11] The main group was in California, and we had part of that same department (under the same manager) in Kingston, New York. (We liked to consider it to be Distributed Intelligence, but some may disagree.)
- Later, I worked in APL development at Morgan Stanley on the Aplus^[12,13] project, which is their own in-house version of APL.
- And so that I don't misrepresent myself, let me point out that I was not one of the APL implementers— I didn't work on the internal code. I focused mainly on the language itself, and attended many of the design meetings and tried to argue for what I believed in regarding language design. My main function there was APL programming and APL documentation.

So in putting this together, one problem that I had was, *where do I draw the line* on history discussions? It's hard to decide whether or not to include some of the APL *applications*, because there have been countless thousands of them, some of which are truly notable. I'll list just a few of them here and then not cover them further in the rest of the more detailed presentation:

- Program trading: We all hear about how the Wall Street people have program trading, where the buying and selling is done automatically, and most of that has been done with APL.
- Insurance companies have traditionally used APL very heavily, to the point where, as I visited insurance customers, I was surprised to discover that many of the actuaries thought that APL was the Actuarial Programming Language, because it was so well suited to what they needed that they assumed it was designed for them.
- Automated warehouse: Chuck Norcutt, a coworker in my department, created the code to keep track all of the products in a major IBM warehouse. This project, written entirely in APL, ran a large robotically-automated warehouse in Raleigh, North Carolina.
- IBM mainframe configurator: Ordering a mainframe computer is a very complicated process. There are a lot of options, many of which are mutually exclusive. So for decades, that was all being handled by a complex set of APL functions.

I understand that at one point they had a team of 125 people working on rewriting it in a different language— and they failed because it was just too complex.

- Space shuttle: The landing of the NASA space shuttle was being calculated for a while by APL; the trajectories were being computed with APL functions from the IBM Federal Systems Division.

And while any of these topics and many others like them might be interesting (and each could be a presentation of its own), I'll try to focus this discussion on areas where the power of the language or unique features within the language have made it stand out, and places where other languages and processes have emulated what APL did, rather than simply discussing applications, no matter how interesting they are.

A deciding point regarding material to be included is: Does this cause you to say, "I didn't know that APL was involved with *that!*"?



I have broken this presentation into two sections, the first being "***Places Where APL was Early to the Game.***" By this, I mean that we will discuss approaches or technologies which APL *did not* invent, but in which APL was used to good advantage early on, accomplishing things that other tools didn't or couldn't accomplish in those days. And the second section will be "***Things that APL Pioneered,***" for those things that actually *did* originate with APL.

▶ Section 1: Places Where APL was “Early to the Game”

■ Instant Messaging

Although I am certainly into technology, I was somehow a late adopter when it came to smart phones. My wife and I finally traded in our flip phones for smart phones just two years ago — and at that point, our son laughed at us and thought, well, they’re finally getting into the 21st Century. “*So I can actually send you a ‘text’ now, right? Do you even know what that is? ... You’re almost starting to get into the modern age.*” And I said, “*Ian, I’ve been using instant messaging since 1971.*” (A blank stare from our son at this point, basically implying, *That’s back when the dinosaurs roamed the earth, isn’t it?*)

This is what really got me thinking about all of the things that APLers have used for decades, and which we may have forgotten were APL facilities years ago. That was the basis for starting this paper.

Messages used to be handled by an APL system command and that was, by any other name, instant messaging... or if you prefer, text messaging. Messages were limited to 120 characters. As an example, let’s send a note to “Bob,” where 265 is the port number that he is on (which I would get from the “)PORTS” command):

```
)MSG 265 HI, BOB. READY FOR LUNCH?  
SENT
```

His response might be:

```
265: OKAY. BTW, I ONLY HAVE 20 MIN. :-(  
:-(
```

I want to point out also that we used these abbreviations, such as “BTW” for “by the way” —none of this is really new— we used that back in the mid- to late-1970s, and that smiley face, or the frowning face in this case, :-(
...we used all this back then.

And the fact that we were sending those text messages around the world back then brought up another important aspect of this:

■ International PTT operations

PTT is “Postal, Telegraph, and Telephone” service, and it is the agency within governments around the world that controls communications. These days, we are very used to the idea of just taking a cable and plugging it into the wall — Ethernet or coax or whatever you have, and talking to the whole world — and it works so smoothly that we don’t think about it. But that wasn’t always the case. PTT has been a tightly-controlled monopoly in most countries, and you could not simply set up your own communication across borders.

Now, in the mid-1960s, along comes APL with this nifty built-in messaging command, and we want to support users around the world. How do we do this? A company which was instrumental in changing how PTT’s handled messaging like this was I. P. Sharp Associates (IPSA) in Toronto, and my hat is off to them for taking on (and winning!) battles at the country level to make sure that they could have a message command in APL and send messages back and forth.^[14] This all predated email, of course, so they did groundbreaking work on setting up a world-wide network, which included messaging.

■ Packet switching

The next part of this is that the I. P. Sharp APL network itself was groundbreaking in several ways. They developed one of the early packet-switched networks.^[15,16] This is simply a faster, more efficient, and more reliable means of sending data around the world. Their semi-private network was officially called IPSANET,^[17,18] and it became operational in May of 1976. This was implemented by Roger Moore, who was also one of the original APL implementers at IBM. So although APL didn't invent packet switching, it was really one of the very first users of that technology on a grand scale.

■ Interactivity

Another part of APL's power is its interactivity. To get into that, let me first discuss the *Grace Murray Hopper Award*.^[19,20] This award is named for Rear Admiral Grace Hopper^[21] and it has been given out since 1971 by ACM (the Association for Computing Machinery). The award goes to a computer professional who makes a single significant technical or service contribution before or at the age of 35. This is a very prestigious award.

Dick Lathwell, Larry Breed, and Roger Moore each received the Grace Murray Hopper Award in 1973.^[22] But the thing I want to point out here is that it was *not* given to them for the APL language — there was no discussion in their award that congratulates them for the notation. Instead, it was presented for “*setting new standards in simplicity, efficiency, reliability and response time for interactive systems.*”^[23,24] We

sometimes forget— everything is so interactive these days, we are used to hopping on a computer and typing something and immediately getting a response or going onto a website and doing the same thing. But it wasn't always that way, and a lot of the push in getting things to be interactive started with APL. So as with other items in this section, APL didn't invent that, but it was very early to the game.

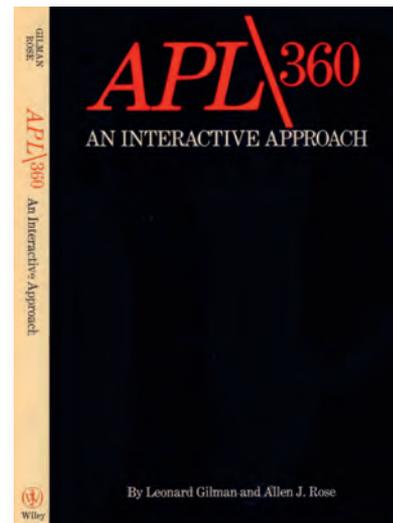


Figure: An early Gilman and Rose textbook

A tribute to the new approach was the Gilman and Rose textbook,^[25] one of the early textbooks on APL (1970). It was named “*APL\360: An Interactive Approach*,” because interactivity was the new and exciting focus at that time, replacing batch operations, so in many ways like this, APL helped to usher in the new era of timesharing.

■ APL helped to usher in the era of timesharing

In the mid-1960s, one of the early timesharing users was NASA. APL sounded like a good tool for them, and they were eager to start using it, but at the time, they had no

way to even dial into a computer system, and back then, the phone companies didn't have much expertise in helping people with data questions, and had no modems to provide to customers. So I was told that it was Adin Falkoff who drove 250 miles from IBM Yorktown to the NASA Goddard Center near Washington with ten Bell System 103A2 telephone data set modems so that they could get on-line with APL:^[26]



Figure: Phone company-supplied modems, c. 1962

APL provided timesharing in an era when many people hadn't yet heard of the concept. In many areas, APL introduced people to timesharing.

■ APL public libraries

Years ago, we became used to the idea that the "APL public libraries" offered repositories of a wide variety of programs and information. Of course, many other systems that any of us have worked with have code repositories, and places where you can go to get help information and so forth. But the APL public libraries always went *way* beyond that, offering not only code libraries with routines such as Fast Fourier Transform and Statistical Analysis, but also

informational sections such as on-line phone books, ordering information for publications, news, email and more— just a lot of things that are *unrelated* to the computer system you're using. Public Library workspaces were created on a huge range of topics. The Syracuse University Computing Center had once commented that "*the sheer volume of material in this library system was overwhelming.*"^[27]

Eventually—*much* later—this whole concept and the work that so many people had put into creating these varied facilities ended up evolving into... *websites*.

■ Election coverage

Election coverage wasn't an APL first; Univac had done this before, but it's another example of a place where APL was early in getting into the game. CBS got election tallying via a dial-up connection into an APL timesharing system.



Figure: Election coverage with APL

I suspect that it was probably still back to that early system in Philadelphia that the APL Design Group used, and one of the stories about this is that Walter Cronkite

was delivering election results at one point, and then he sort of stopped, and there was some confusion— back in those days television news didn't have the flow that it has today, so when a problem came up, they would sort of stop and look around. The problem was that Walter Cronkite had accidentally kicked the modem under the table with his foot and the phone line was disconnected. So as he was trying to fill time, one of the technicians was crawling in under the table in front of him to dial up the APL system again and get it back on the air so that he could get some more election results. ...“*And that's the way it is.*”

But even with an occasional glitch like this, the point is that APL was very early in helping to bring this kind of information to people.

■ Early public computer demonstration

An early goal was just getting to show the public what a computer is all about, and this is a place where APL was involved very early.



Figure: IBM 2741 Selectric Terminal

A computer demonstration was set up at the Franklin Institute Science Museum in Philadelphia, consisting of an IBM 2741 (Selectric typewriter-style terminal)^[9,10] connected to the APL Design Group's machine in the middle of Philadelphia — and that's probably the first view that people in Philadelphia had of what a computer was all about. They could actually get onto the 2741 and type in APL expressions, so of course this was all *hands-on*. Probably these days we wouldn't let people use a live connection into a development system, but somehow it all seemed to work back then.

■ Computer Viruses

Do you remember the old “self-replicating APL expressions”?^[28] The challenge was, what is the shortest piece of APL code you can type that will return *itself* — an expression that would return exactly what you typed? It has to have some computing to it; it has to have at least one function, so you can't just type “4”, for instance. Here is an example of an expression that returns itself:

```
22ρ1ϕ11ρ' ' '22ρ1ϕ11ρ' ' '
22ρ1ϕ11ρ' ' '22ρ1ϕ11ρ' ' '
```

Okay, there's no practical point to it, just some fun, and perhaps it's a learning tool, but as a self-replicating expression... well, a variant of that —perhaps a distant cousin of that (...or maybe its evil twin)— is a *virus*. So when did the *first* computer virus get created?

I read through a lot of articles on viruses and one from the BBC talked about what they claimed was the first computer virus,

and it dates it at 1983.^[29] Another article struck me as being interesting, which said that in 1982, a student named Rich Skrenta at Mount Lebanon High School in Pittsburgh, Pennsylvania wrote the “Elk Cloner” virus^[30]— that caught my attention because that was my high school (...but no, I didn’t write any viruses). Skrenta was in 9th grade at the time and he created a virus that was originally supposed to be a practical joke, but it got out of hand, and that was what *this* article claimed was the first virus, in 1982.

However, at a meeting of IBM’s APL ITL (Interdivisional Technical Liaison) Committee^[31] in California in 1976 or 1978, we had discussions of computer viruses — and that was a new term to us at the time. Larry Smith was an IBM executive who had been appointed to study this new phenomenon, and he came to our conference and talked about viruses. He surprised us by pointing out that the first computer virus was written in APL.

Now, I am pleased that I don’t have to report that this virus was damaging — it was not. He emphasized that this was a *beneficial* virus. (...A what?) As he described it, this procedure could replicate corrections to other machines and it was therefore described as being analogous to a *flu shot*, delivering a carefully-controlled virus to *protect* you. Obviously, the correction itself had to be correct and he pointed out that when they realized that other viruses written in any language had the potential to do harm, they really started studying this, and he was appointed to be in charge of that study. These discussions precede other virus articles that we read about elsewhere.

■ Email

Although there are some differing opinions as to when the first email system appeared, credit has at times been given to an MIT in-house system, which was developed in 1965.^[32] This was a very limited system, intended for use just within their own campus. The other contender is a system created by Frank Bates III of Mobility Systems, written in 1971.^[34] These were both very limited systems, with neither one handling any commercial usage.

A discussion of “Internet in Its Infancy”^[32] tells us that “*By the end of 1971 there were 23 computers at 15 different locations connected to the ARPANET ... The following year [1972] the first true email software was written, and email rapidly became the most popular application on the network.*” This first “*true*” email system was created by Larry Breed, one of the founders of Scientific Time Sharing Corporation (STSC) in Bethesda, Maryland. He created it in 1972, and it was of course implemented in APL. This very successful email system was described as the “APL*PLUS Message Processing System Mailbox,” or known to their customers simply as “the Mailbox.”^[33]

In that same Internet history article, the following statements were made about the STSC mailbox:

“In 1976, email was first used to gain political power. Jimmy Carter and Walter Mondale used email during their U. S. presidential campaign to co-ordinate their schedules. They won the election and Carter became a strong supporter of the internet. Each of their emails cost about US\$4 to send.”^[32]

White House Press Secretary Jody Powell talked about how they used the STSC Mailbox to help them with Jimmy Carter's successful 1976 U. S. Presidential campaign, saying:

“One of the constant problems in a political campaign is caused by the fact that things happen in rapid succession. The candidate tends to be one place, most information —the ‘good’ information— tends to be another. We of course used the Mailbox system to get that information and have it available when the candidate and travel party needed it. It saved us some time, it saved us some money ... I think you'd have to say, all in all, that it worked out pretty well.”^[34]

This same email system was also used by I. P. Sharp Associates (IPSA) in Toronto; these two “friendly-rival” APL timesharing companies shared the same code for their email. Leslie Goldsmith at Sharp rewrote it to incorporate greater security. That new facility was then known to the Sharp APL customers as “666 BOX”^[34,35]

So, with just one or two very limited, in-house systems preceding these early APL mail systems, I can't *quite* claim that APL actually “invented” email, but in a practical sense, maybe that is the case: APL systems offered their users the first “real,” widespread, global email service. And all of this was available to APL users before the term “e-mail” had even been coined.^[36]

I had purchased a couple of APL terminals to use at home (...yes, terminals, not PCs). Starting in the mid-1970s, I was a guest user of the Sharp APL system, and had an

email account there, and I thought, *this is terrific* —we should have something like this. So although I never saw any of their code, I created an email system^[37] at IBM based on the ideas that I got from the I.P. Sharp email system.

Email was still quite rare in these days, so following the Sharp system, mine ended up also being another *one* of the first email systems (but obviously *not* the first)— and of course, all written APL.

Now, since I am not an authority on the STSC or IPSA mail systems, let me tell you about the follow-on system that I created.

In support of our timesharing system, I had a great manager at IBM, Bill Davis, who gave me the freedom to work on the projects that I felt needed to be done, so having seen the IPSA mail system, I decided email would be a good thing to have.

It started as a personal project in 1979, and I released it to the world in the summer of 1981... and by “the world” I mean that on Day One, our users in seventeen countries had email. They were all running on our mainframe computers there in Kingston, New York, so we had people in Japan, Australia, Switzerland, Germany, France, England, Argentina, Brazil —all around the world— logging on to the Kingston system to do their daily work, and now for the first time, also to get their email.

Before I put my email system together, I went to Toronto and met with Leslie Goldsmith to ask him what I should be really focusing on if I create my own email system.

He said the *Number One Priority* needs to be *security*— he said that they have competing companies on their system, all using email, and they have to know that it is secure, that nobody else can see their email. So that was one thing that I made a big point of with my system, to ensure that everything was very secure and that nothing could get misdelivered. There were even some new features added to the APL interpreter itself to ensure the security of the messages.

Now, flash forward to today and look at our current email providers: How secure is email these days? If you were signing up for this conference, you might go on to a website with your credit card number to reserve a hotel— that may be fine. But would you send it by email? I would suggest not.

A lot of different *groups* within IBM used our email system, but I was particularly interested in hearing that IBM Headquarters in Armonk became interested in this system and adopted it as their method of sending out all of the IBM Corporate Communications announcements around the world. So all of the new product announcements and executive promotions and IBM earnings statements, and so forth —everything that went up on the (physical) IBM bulletin boards worldwide— came through my email system.

All of these various APL-based email systems predated CompuServe, just in case anyone still remembers that (limited email in 1989 and full support in 1992), AOL email (1991), and Yahoo (1997). It greatly predated Google (1998) and Google's Gmail

(2004), and of course it predated personal computers.

It also predated the commercial Internet (early 1990s). Sharp had their own IPSANET; so how did I send email around the world at IBM? Well, IBM had its own global network, called VNET^[38] (mid-1970s), some portions of which still exist, but of course the Internet has taken over for a lot of that now.

An attempt to standardize email formats came in September of 1973 from the Internet Engineering Task Force (IETF),^[39] but it was then nearly a decade until SMTP (Simple Mail Transfer Protocol) was introduced as the standardized format for email messages, using the now-familiar “@”-symbol in email addresses (August 1982).^[40]

All of this came a decade after many APL users had already been using global email.

I'll emphasize again that of course I realize that I was certainly not the first one to create an email system; I don't claim to be, and I don't even claim to be the first one to create it in APL code. I commend Larry Breed and the others who got the APL world online with email.

The point of this discussion is that APL was very early to the game with email. In fact, it was early enough that with the first release of my system I had to explain to people what “email” was. Some people said to me, “*Of course I know what 'mail' is, but what is the 'e' part? ...Does that somehow make it different?*”

Yes... yes, it does....

■ **Advanced DNA analysis**

Getting into more current topics now, techniques for DNA analysis have been done since the mid-1980s by many people, using many different tools and languages. But imagine what could be accomplished if we were to bring in a person who knows both DNA *and* APL.

Charles Brenner is a renowned expert in DNA analysis and in APL. In his words, he “*leverages the nimbleness of APL to identify criminals, fathers, World Trade Center and tsunami victims, and determine race using DNA in a world of fast-changing DNA identification technology.*” He speaks of his DNA works as “*an application tailor-made for APL.*”

In April 2013 he entered into a competition through NIST (National Institute of Standards and Technology). Over a hundred entrants competed. Five problems were presented, each having difficult situations for DNA analysis.

In his words, “*One of the competitors (supported by years and millions of dollars of government grants) got four problems right and came close on the fifth, viewing it as a three-person mixture, but in fact it was four.*”

Brenner alone correctly analyzed all five exercises, and furthermore correctly diagnosed one suspect as a mixed race person.

Brenner is doing this with Dyalog APL.^[41]

He said, “*We APLers try to be modest but it’s not always easy. Sometimes there’s just not much to be modest about.*”^[42-46]

▶ **Section 2: Things that APL Pioneered**

■ **Uniformity of math notation**

Iverson used to point out that when people say that they had a hard time with math in school, the all-too-common comment is that their math teachers must not have been very good. Ken has suggested that this may be misplaced blame; part of the blame should be on the notation itself, which is saddled with different rules for so many different operations:

a)	-5	-5
b)	$5!$	$!5$
c)	$ 5 $	$ 5$
d)	$\sum_{i=0}^n$	$+/n$
e)	$16 \overline{)44R7}$	$16 711$
f)	$\frac{3}{5}$ $3/5$ $3 \div 5$ $5 \overline{)3}$	$3 \div 5$

In conventional math notation, sometimes the function has to be on the left (a); sometimes it’s on the right (b). Sometimes it’s on both sides (c). Sometimes we have symbols that are scattered around each other (d). Sometimes we have just one symbol, but it goes part way around the numbers and semi-encloses it... and there may not even be an official symbol for the result that we actually want; in this case, the remainder (e). And for *something as simple as division*, why isn’t there one standard way to write it? We have several totally different ways of writing it, and all of them are in very common usage (f). Ken thought this could be improved, so he provided a means of making math notation much more uniform, and even apart from programming —just for someone learning math— I think that this

is a wonderful step forward, and I'm hoping that more attention can be put to this in the future, just as part of basic learning in schools.

■ **Introduction of new symbols**

It was pointed out that Ken Iverson is the only mathematician in history to put more than two new symbols into common usage—that is, of course, assuming that we consider APL to be “in common usage.” It was Donald McIntyre who made that observation, in his paper, “*From Hieroglyphics to APL*”.^[47,48]

Confucius has told us that “*Signs and symbols rule the world, not words nor laws.*” (Okay, although I have often seen this quoted, I can't actually vouch for its authenticity. ...It might be apocryphal, but it *is* food for thought.)

■ **International language**

I see APL as being the only truly *international* programming language. (I am considering only serious work-related languages, not toy languages, and I am grouping APL's derivative languages, such as J and K, with APL.)

For an expression that I write for finding unique values (shown below), *I* is unique (...I try to tell people that, by the way...), then if I transfer this code to a user in another country, he should be able to read the code just as easily. A problem with other languages is, because English has been used

so commonly for programming languages, if you are in France or Germany and want to do some programming, often you need to learn some English first so that you know the *keywords* in the operations. And if you are in Greece or Russia or Japan or China, for instance, you may have to gain some familiarity with the Latin alphabet, which we use here, in order to use a programming language. Not so with APL.

And sure, although English is widely used for technical work of all kinds, not *all* programming languages use English; there are languages that are localized for use in other countries, of course — but then they are also trapped in *that* language. APL makes it much simpler — the code should always be the same; only the comments and the object names need to be in local languages.

With APL, whether I'm showing it to a Spanish audience or French audience or Japanese audience, we have *no keywords*, so the code is going to be the same; the comments may differ, but the code itself should always be the same. So I maintain that it's still the only truly international language.

You might say, ah, but how about if I get an error message... that's in English, isn't it? What do we do about that? Under APL2, National Language Translation^[49] lets you specify what national language you prefer, and that's what is used for error messages:

$I \leftarrow ((V \uparrow V) = \uparrow \rho V) / V$	Ⓐ <i>Find unique values</i>	(English)
$I \leftarrow ((V \uparrow V) = \uparrow \rho V) / V$	Ⓐ <i>Buscar valores únicos</i>	(Spanish)
$I \leftarrow ((V \uparrow V) = \uparrow \rho V) / V$	Ⓐ <i>Trouver des valeurs uniques</i>	(French)
$I \leftarrow ((V \uparrow V) = \uparrow \rho V) / V$	Ⓐ 一意の値を見つける	(Japanese)

Figure: APL coding with different national languages

```

      1 2 3 + 4 5
LENGTH ERROR
      1 2 3+4 5
      ^      ^

      □NLT← 'DEUTSCH'

      1 2 3 + 4 5
LAENGENFEHLER
      1 2 3+4 5
      ^      ^

```

System commands also can be entered in your national language and the responses from the system commands of course will come back in that language. It was released in over a dozen languages: Danish, English, Finnish, French, Canadian French, German, Hebrew, Italian, Japanese (double-byte), Katakana (single byte), Norwegian, Portuguese, Spanish, and Swedish— with more added later. And Help text is also in your selected language.

Here’s just a bit of trivia: What do you think the very *first* national languages were that APL2 supported? Each of the languages that were released were specified by knowledgeable representatives from each of those countries, so they were done properly (not just done by code developers sitting down with a dictionary and trying and translate things themselves). Those of us in the development group aren’t the experts in those various national languages, so *just for testing*, we had to have something else, and therefore the first alternate languages to be implemented were TexMex and Pig Latin... with TexMex being things like, “*WORKSPACE ESAVED*.”

■ Array processing

Some other languages support arrays, but most commonly just for storage or for some selected operations, and I guess I don’t need to tell this audience, but having everything work as array operations was a huge pioneering step for APL. No one else has truly caught up yet.

■ First desktop Personal Computer

In discussing the first personal computer, some definitions need to be established, because the term “personal computer” hadn’t even been used yet when the first of the machines that we’ll discuss here were created. So let’s define a personal computer to be a desktop (or portable) computer that’s primarily intended for one person to use, rather than shared use. These machines were commonly used in a home, although small businesses of course also found them to be a great tool.

In defining what a “personal computer” is, IBM’s early-1970s *mainframe* offering called VSPC (“Virtual Storage Personal Computing”) doesn’t help to clarify matters. (As an aside, VSPC did support APL, but this was a mainframe offering, not a personal computer.)

So, what was the *first* desktop PC? The Apple II is commonly given the credit as being the first desktop computer—but *was it*? I started looking into that, and as a starting point, let’s look at the Apple I computer,^[50,51] which was introduced in 1976.

The Apple I wasn’t actually a full computer. It was sold as a hand-built circuit board—just the board— at the Homebrew Computer

Club in Silicon Valley, California.^[52,53,54] Steve Wozniak built each of these boards himself, by hand.^[55] The Apple I lacked a keyboard, monitor, and storage, so in this first example, somebody got his own keyboard and built a wooden case for it and so forth:



Figure: Apple I circuit board in a wooden case

And here's an example of some work done by a person who put their circuit board into a briefcase, to configure it as a portable computer:



Figure: An Apple I circuit board in a briefcase

But again, this first offering was just a circuit board, not a complete computer. Then, the following year, the Apple II computer came out.^[56-59] This was a complete, ready-to-run computer, and it was *said* to be the first desktop personal computer... but again, *was it?*



Figure: Apple II computer

Actually, the first personal desktop computer was the IBM SCAMP machine, built in 1973.^[60] “SCAMP” meant “Special Computer APL Machine Portable.” This predated other personal computers, and it ran *only* APL. It may look vaguely familiar to some of you:



Figure: The IBM SCAMP machine, open and closed

This was the SCAMP and following are some excerpts^[61] from IBM management as they donated the first SCAMP computer to the Smithsonian Institution in Washington, DC, where it remained on display for years. Following are statements made by Paul Friedl, the manager in charge of the development of the SCAMP:

“One of the reasons that I am pushing to exhibit SCAMP is that we want everybody to know —contrary to popular opinion— that the micro-computer was not born in January 1975, and it was not a technology that could only be done by teenagers in garages.

“So how did the innovation of SCAMP come about, especially within IBM, which at that time was largely organized around the concept of large centralized computers and timesharing terminals? It happened because IBM had previously created two laboratories in California’s Silicon Valley. Both of these labs were given free rein to

seek and develop new systems and business opportunities for use within several years. These two facilities formed the ideal environment for creating a revolutionary device like SCAMP, the first IBM personal computer. All that was needed was a direction.

*“The direction came in 1972 as Paul Friedl —that’s me— a manager in PASC [Palo Alto Scientific Center], conceived the idea for developing SCAMP, a personal, portable IBM computer. To prove feasibility for this idea, I presented IBM executives with an ambitious plan to build SCAMP within six months, and demonstrate it on the desks of IBM executives. At that time, there were no Apple or Microsoft Corporations, and the term “personal computer” had not yet entered into the popular lingo of computing. Since 1972 was a highwater-mark in centralized mainframe computing, when the IBM execs heard my plan, they were stunned, and after a few seconds they spoke out: ‘**Wouldn’t that be something!**’ And that phrase became our project motto. ...*

*“Created by Ken Iverson, APL is a beautiful and extremely powerful programming language which was in use throughout IBM in the 1960s and ’70s. APL could do almost anything, but even more importantly, **APL changed the way you thought.**”*

So there are two myths which the SCAMP developers sought to debunk: The first personal computer was not created by two teenagers in their garage; it was created by IBM. And the first personal computer didn’t run BASIC; it ran *only* APL.

PC Magazine called the IBM SCAMP a “revolutionary” concept and “*the world’s first personal computer.*” And now you might think, this was 1973 and they didn’t know what was coming next month or next year, and they didn’t really have a perspective on what might be perceived as being revolutionary later on — well, I want to point out, this was a statement they made in their 1983 issue^[62] —ten years later— so they had plenty of perspective, and certainly recognized it as being the first personal computer, and indeed, revolutionary.

After the SCAMP prototype, but before the IBM PC, and for that matter also *before* the Apple II was the IBM 5100.^[63,64]



Figures: The IBM 5100 Computer



The 5100 could be purchased as a complete system, with an external tape drives and a printer. Below, here’s an ad showing a strong man,

holding his breath and grunting while he is trying to hold this 50-pound machine up in the air just long enough for the photo to be taken:



Figure: Advertisement for the IBM 5100

This magazine ad^[65] said, “Now you can have a computer right on your desk.” That had never been done before. It features “a typewriter-like keyboard” (you probably wouldn’t sell a lot of machines today by telling people that); “a 1024-character display screen” (which is another way of saying that it’s all character-based, so there’s no graphics); “integrated tape drive” (again, not a big selling point today, but certainly was then); and “16k characters of memory, expandable up to a maximum of 64k characters.” APL could easily have supported more memory, and the designers wanted to provide that for APL, but the Fortran implementation that they were using only supported 64k, and the planners insisted that they be kept equal. Too bad.

The ad goes on: “Also available is a communications feature, which allows a 5100 to be used as a terminal.” Our department got two of these machines when they came out,

and they were wonderful as terminals. One of the nice things was that we weren’t limited to 134 baud(!) anymore on the 2741 dial-up lines; now we could go to the *high-speed* dial-up lines and run this baby at 300 baud— pretty amazing... for its day. It was released in three models:

- Model A was APL only
- Model B was Basic only
- Model C was a Combination machine: APL and Basic

And as an aside, just for fun, do a Google search sometime for IBM 5100 and John Titor.^[66] He has been discussed on many forums as a time traveler from 2036 who has gone back to 1975 to get an IBM 5100 with APL to solve problems in 2036. And apparently they also want that highly-coveted 370 emulator that let them run APL.SV on the 5100. (...If he is at this meeting, by the way—as he should be— please let me know; there are a few things I’d like to discuss with him...).

At 50 pounds, the 5100 may seem pretty heavy by today’s standards, but remember that it replaced what would previously have been *half a ton* of equipment. Its predecessor was the IBM/1130.^[67,68]

And no, that’s not on a desk, it *is* the desk:



Figures: Versions of the IBM/1130

The machine on the left is an 1130 with the *expanded memory* installed: There's an additional three-foot cube bolted onto the left side, giving you an extra 8k of memory—good for its time. And yes, it ran APL. The first time I saw APL running was on an 1130:

```

)33
KYM SIGNED ON

                A P L \ 1 1 3 0

5050          +/i100
              (i5)°.+i5

      2      3      4      5      6
      3      4      5      6      7
      4      5      6      7      8
      5      6      7      8      9
      6      7      8      9     10

)OFF
SIGNED OFF
    
```

Figure: A brief sample APL session on an IBM/1130

It was Larry Breed and Charles Brenner who ported APL to the IBM/1130.^[69]



There were a few other machines which need to be included in these discussions. A Canadian company called Micro Computer Machines brought out the MCM/70 computer; it was demonstrated in 1973, but not available until the end of 1974.^[70-75] It had an odd one-line plasma display, and was further notable in that it ran *only* APL.



Figures: The MCM/70 APL computer

Although it was predated by IBM SCAMP machine, due to the limited availability of the SCAMP, the MCM/70 can make a reasonable claim as being the first commercially-available personal computer.

Although it was meant to be plugged in, the MCM/70 was also unusual in that it had a built-in battery, likely the first PC to offer that. So even if you somehow didn't consider it to be the first PC, it can certainly claim the title of "the first truly portable computer."^[76]

This brought about another first: Zbigniew Stachniak, the developer of the MCM/70, told this story:^[77]

"In August 1973, MCM sent Ted Edwards to the APL Congress in Copenhagen with a prototype of the MCM/70. What was

unusual about that MCM/70 was that it was mounted in an attaché case and was operating on batteries. Edwards was not only able to board the plane with this unusual device but also reviewed his presentation using that MCM/70 during the flight to Copenhagen. This constitutes another 'first' for MCM: the first portable computer operated during a flight. And, of course, that 'laptop' was running MCM/APL. The MCM/70 story was picked up by the Danish daily Politiken on 1973-09-28."

As an aside, if you haven't read the stories on Roger Hui's "APL Quotations and Anecdotes" webpage from Jsoftware,^[78] I encourage you to do so. There is a wonderful collection of interesting and informative APL stories there.

The MCM/70 also broke ground in another area: Back in 1975, this was the machine that introduced APL to the Soviet Union. The Computing Center of the Academy of Sciences of the USSR purchased several of the MCM/70 machines.^[79-82]

Regarding other machines which have sometimes been called "the first PC," you might hear that the Altair 8800^[83,84] was the first PC. It actually came a few months later, in 1975:



And then there was the intriguing Ampere WS-1, an APL laptop produced by Nippon-Shingo in Japan.^[85-89] Although it isn't a contender for the first PC, it was a very early laptop— probably even predating that *term*, as they called it a "knee-top" machine. This very snazzy-looking *APL-only* laptop was released in 1985, with all of the sexy, curvaceous styling of a sleek sports car:



Figures: Ampere WS-1 APL laptop

And there's a reason for that: The Ampere WS-1 APL laptop was designed by Kumeo Tamura, who also designed the exterior lines of the *iconic* 1970 Datsun 240z sports car.^[90,91]

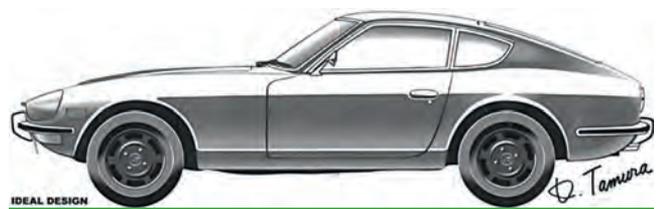


Figure: Design concept for the 1970 Datsun 240z

...So, sure, might as well make use of that design a second time.

Unfortunately, the Ampere WS-1 failed its FCC (Federal Communications Commission) certification for the U. S. marketplace, so it was never able to be sold in the United States.

So, regarding the first desktop PCs, here was the progression:

Timeline:

- IBM SCAMP — APL only . . . July 1973
- MCM/70 — APL only Nov 1974
- Altair 8800 Jan 1975
- IBM 5100 — APL and Basic . Sept 1975
- Apple I circuit board July 1976
- Commodore PET Jan 1977
- Apple II computer Apr 1977
- Radio Shack TRS-80 Aug 1977
- IBM PC (5150). Aug 1981
- Ampere WS-1 — APL only. . Nov 1985



All in all, APL played a larger role in the early PCs than you might have expected. So why wasn't APL considered as the default language for PCs? Well, to some extent, it was.

Prior to forming Microsoft, Bill Gates had already known about APL. Gates met with Ian Sharp, Eric Iverson, and Bob Bernecky at IPSA in the very early days of PCs, and talked about using APL.^[92] In February of 1976, Bill Gates announced via “*An Open Letter to Hobbyists*” that Micro-Soft (as it was named back then) was working on “*writing 8080 APL and 6800 APL.*”^[93] According to Zbigniew Stachniak from MCM, Gates stated that “*Equivalence with the 5100 was my goal.*”^[94] Stachniak further stated, “*It was not until 1979 that Microsoft announced its APL-80 interpreter for the Intel 8080 and Zilog Z80 platforms. It was to be out in April 1979 and compatible with IBM’s APL.SV software. But in the end the Microsoft APL-80 proved to be vaporware and by the early 1980s, it was Microsoft’s BASIC and not APL*

that was installed on the majority of personal computers.” So in the end, the Microsoft APL products never saw the light of day.

This is reminiscent of the VHS-versus-Beta format wars. We can argue about which system is better, but there is no doubt that marketing plays a very important role in public acceptance.

Personally, I think that it's a *good* thing that Microsoft *didn't* pursue APL further— because if they had created their own version of APL, I have to think that it might have ended up being vastly altered from what we are accustomed to today, and due to volume, might have become the accepted standard for APL. I am pleased with the APL offerings and continued development that we are seeing these days, from multiple companies.

■ Early pocket calculators

Going down to the very small end, in about 1972, at a time when pocket calculators were very new, Walt Niehoff at IBM in Endicott, New York worked on creating a handheld APL calculator. (To steal a line from the SCAMP folks, “*Wouldn't that be something!*”) I remember seeing the breadboarded prototype for this, which filled several feet of a rack cabinet in his lab in Endicott. But I'm sorry to report that it did not get released as a product.

Just as an aside, the HP-35 calculator,^[95] launched in early 1972, was the first pocket calculator to be released with scientific functions, able to replace a slide rule; other calculators of the time had only four functions. I am told that Hewlett Packard designed that calculator using APL\1130.^[96]

■ **Supercomputers**

At the other end of the scale are the supercomputers. So *what is* a supercomputer, and what *drives* it? In the mid-1980s, IBM viewed a supercomputer as being their top-end mainframe machines which were also fitted with the IBM Vector Facility;^[97,98,99] which turned their largest mainframe into a supercomputer. The Vector Facility was supported only by Fortran and APL, and that made APL very popular with IBM. And it was our *Numerically Intensive Computing* Group in Kingston that created that.

And one of the really beautiful points about using APL with a Vector Facility is that no one's code has to change at all — no tweaks, no changes, and no recompiling... come Monday morning, even old code just runs *faster*. Fortran couldn't offer that.

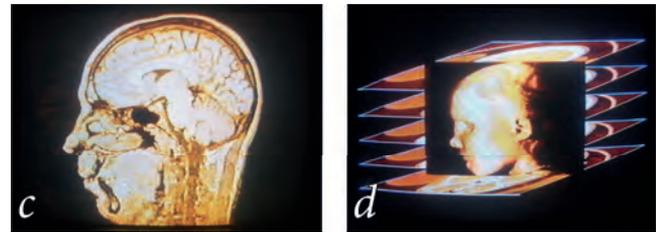
■ **APL in Cancer Research**

In 1989, Prof. Dr. Hans-Peter Meinzer, at the German cancer-research center DKFZ ("Deutsches Krebsforschungszentrum") in Heidelberg, Germany, analyzed Magnetic Resonance Images (MRI) and Computer Tomography (CT) scans with APL using an IBM 3090 mainframe with the Vector Facility.

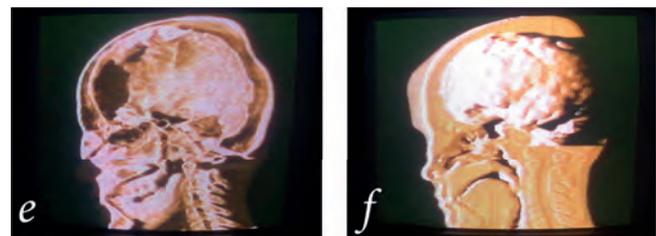


Previously, the many images that are created by the MRI (a) and CT scans (b) had

to be viewed individually (c), but using the array-processing power of APL, Dr. Meinzer was able to produce highly-detailed 3-D medical images (d):



Using sophisticated techniques of anatomic segmentation and semi-transparent raytracing (e), Dr. Meinzer was able to construct 3-D rotating views (f) of a human head of a patient who had a cancerous tumor:



Figures: MRI and CT scans and viewing of images

A demonstration video of this, called "Viewing the Invisible," was shown at the APL90 Conference in Copenhagen. John Mizel, Michael Van Der Meulen, and myself, in IBM's *Numerically-Intensive Computing* group in Kingston, New York collaborated with Dr. Meinzer in creating this video.^[100] (I apologize for the low-quality images shown here; Dr. Meinzer's procedure produced very high-quality images; however, *these* images are the result of viewing a quarter-century-old VHS tape on poor equipment.)

The goal was to provide physicians with better visualization techniques, giving them

the means of seeing details which *might otherwise go undetected until the moment of actual surgery.*

Thanks to the array-handling power of APL and the increased processing power of the Vector Facility, Dr. Meinzer's work far surpassed any medical imaging that had ever been done prior to his work.

■ Complex TV graphics

In the 1980s, all of the major U. S. TV networks suddenly came on the air with fancy, glistening, highly-reflective 3-D floating type and images that fly in from the side with swooshing sounds and so forth.

These days, we are so used to seeing fancy 3-D logos and images that we scarcely notice them anymore, but back in the early 1980s, they were stunning. And they all seemed to arrive at the same time. I remember thinking back then, how is it that all of the television networks have come up with that *simultaneously*?



Figures: Early TV logos and images made with APL

All of these logos appeared at about the same time because they were all created by the same person: It was Judson Rosebush,^[101] using APL and Fortran. His company was Digital Effects, Inc.^[102] He was creating the graphics by actually dialing into the STSC APL system in Bethesda from New York City, and doing it all via dial-up timesharing (at 1200 baud). I was told that one month his timesharing bill hit a quarter of a million dollars, and at that point he reportedly decided that it may be time to get their own mainframe.^[103]

By the way, regarding the spinning globe that was used for the NBC Nightly News intro, a little-known fun-fact is that their first pass of the animation inadvertently had the globe spinning in the wrong direction, so it had to be reworked.^[103]

In addition to creating the network logos with APL, they also created a lot of TV commercials for companies around the world using APL-generated graphics. Digital Effects became so closely associated with this work that its name became a noun for this type of work.^[102] Additionally, Digital Effects did some work on movies; in *Xanadu*, complex scene transitions were created by Judson Rosebush with APL.^[104]

Judson Rosebush's company, Digital Effects, was also one of four companies that collaborated to create the movie *Tron* in 1982.^[105] This movie used APL heavily for creating the visual effects. But although *Tron* was a groundbreaking film, it was never presented with an Oscar for technical achievement or special effects. Why? ... Because at the time, the Academy felt that

Tron had “cheated” by using computers to create the scenes.^[106]

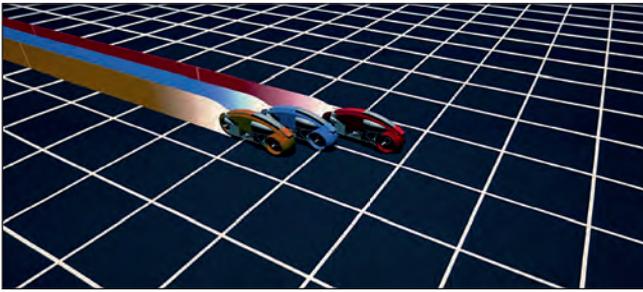


Figure: Graphics from the *TRON* movie

■ Robust word processing

Part of my job has always been to write documentation, so word processing has always been important to me. Finding the tools to do that has been difficult at times. It’s so simple now — we sit down at any PC, and we have Microsoft Word or whatever other tool you want. These days, I have Adobe InDesign on my machine; that’s a *wonderful* tool for text processing and page layout. But before Microsoft Word came out in 1983... before WordPerfect appeared, back in 1979... and even before WordStar and T_EX appeared in 1978 — how did anyone do it? Well, back then there was [fanfare] *The APL Text Machine*.^[107]

The APL Text Machine was originally called APL text (pronounced “appletext”). The version that was first used in Yorktown Heights was designed by Adin Falkoff, and Mordecai (Morty) Zryl was the APL programmer. Later enhancements were added by Elizabeth Llanso. Maintenance was handled by Don Orth starting in 1974.

The APL Text Machine was used to produce the first APL\360 manual in late 1966 or early 1967. Agnes Carlin used it to

produce several of Ken Iverson’s books, e.g., *Elementary Algebra*.

The APL Text Machine was started in the late 1960s, and formalized by the 1970s. It was put out into the APL Public Library at that point, so that any of our users could use it. It really solved a lot of problems for us. With the text machine we could —*for the first time*— have the printer “automatically” switch fonts back and forth. When we were writing APL documentation, we *had* to have that. For other people who were writing purely text documents, perhaps this wasn’t as necessary and you could get by with lesser tools, but we needed something robust.

By the way, I put the word “automatically” in quotes when I spoke of font switching, because of course we were using the IBM Selectric typing element^[108,109,110] (or “type ball”) back then, and so it wasn’t *really* automatic. We had to manually switch the type ball but the point is that *we could do it now* — the APL Text Machine finally provided the commands for controlling it.

Our IBM 3211 system printer was set up with an “APLFULL” print train, which contained the APL font plus an upright caps and lowercase text font, so the APL Text Machine could print on that device without any manual intervention, but the print quality wasn’t good enough for publications; it was really just meant for daily reports. For publications, I chose the Selectric terminal with a high-quality single-pass Mylar ribbon.

As an aside, I went onto the Web to get a couple of pictures of Selectric type balls

for the presentation, and I thought that it was pretty funny that the picture of the APL type ball was *of course* missing a couple of teeth. That seemed to always be the case; remember that? It would then misprint some of the characters because the printer couldn't rotate it properly:

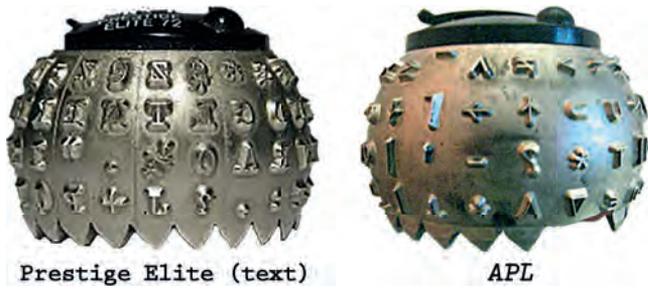
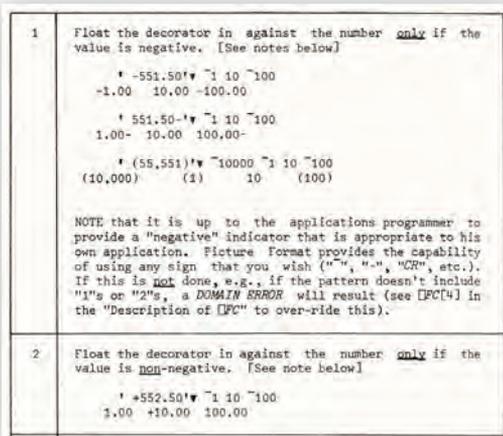


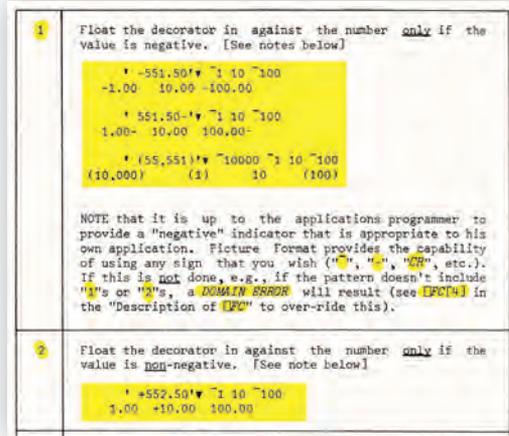
Figure: Selectric type balls

These days, we don't even think about the complexities involved in changing fonts; we just switch fonts back and forth within Word and then print it on pretty much any printer. But it wasn't always that easy.

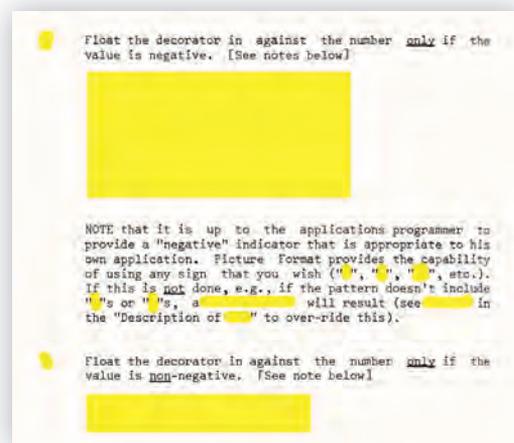


Here's a portion of a page from a newsletter that I put together back in 1980. This was a Reference Issue of *Jot Dot Times*^[111] that showed how to use, in this case, Picture Format. It doesn't look like it would be that hard to type this up, *except* that we realize that we need to switch fonts back and forth — and not just within blocks of text,

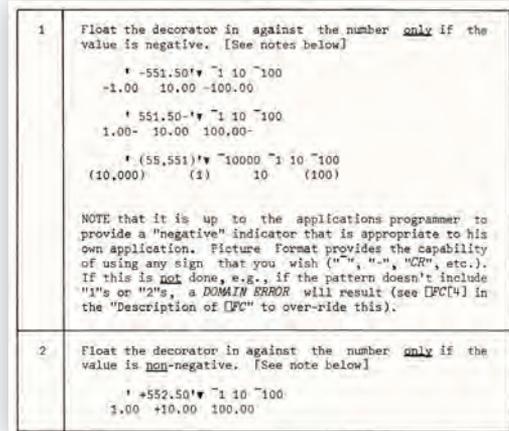
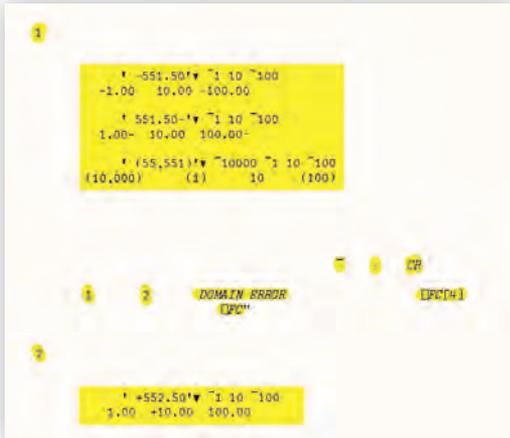
but buried within paragraphs, too, so manual (physical) cut-and-paste was just far too involved to consider:



The highlighted part is in the APL font and the rest is in a text font. The APL Text Machine let me enter codes to indicate all of the formatting that I wanted, including font switching. But having it stop and make me change type balls back and forth, perhaps many times in each paragraph, would be far too tedious, so it had an easier approach: For each *page*, it would print *just the text portion* first...

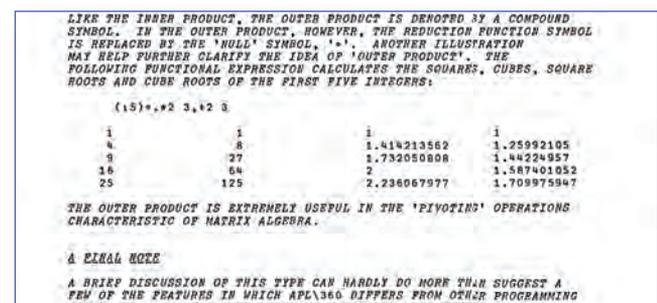
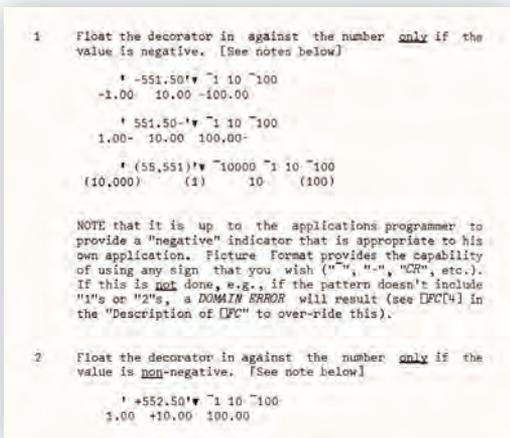


...and I would then roll the page back up to the top, change the type ball, and press Enter, and it would print *just the APL portion* of that page:



At this point you would *hopefully* end up with the two parts lined up. Yes, there were indeed some spoiled pages, but this is a photo of a page from the actual newsletter that I published, so they certainly could be made to line up:

It was quite a task to create APL manuals years ago, but thanks to the APL Text Machine, we finally had the tools to do it. But there were APL manuals before the APL Text Machine was available, so you may ask, how did *they* do it? The answer is: poorly... very poorly:



So, we're done now, right? Well, no, not quite. Unlike more modern products like MS Word, the APL Text Machine didn't have any provision for drawing the lines around tables... *not* because that couldn't be programmed into it, but because back then, we didn't have any *printers* that could draw the lines—everything was character-based. So the lines had to be drawn in manually, very carefully, using a technical-illustrator's pen (sometimes producing, of course, some more spoiled pages):

This is an APL manual from 1960s,^[112] *prior* to the APL Text Machine. Headings were just done with underscored letters and the text was all caps, because the APL type ball didn't have lowercase. Remember that we needed to have text and APL symbols intermixed, and prior to the APL Text Machine, the tools for doing that just didn't exist.

■ **Commands within text**

One of the things that came out of this formatting work is the introduction of *readable commands within text*. The APL Text Machine let you type your text and at the beginning of the line, if you put in a delimiter

character, like a slash, what came after that was a command rather than text. We are very used to that now with a lot of facilities we use, but where did this come from? Well, IBM Script was released in 1970, and it had readable commands within text like that. T_EX was released in 1978, and it is certainly based on this concept (and by the way, I was told that Donald Knuth modeled T_EX in APL^[113]). SGML (Standard General Markup Language) was formalized in 1986, and its cousin, HTML (HyperText Markup Language) was released in 1993. All of these facilities used this approach of embedding commands within text. But before any of these were available, we had the APL Text Machine.

A patent claim was presented to IBM in the 1980s claiming ownership of the concept of embedding readable commands within text. That claim found its way to Adin Falkoff who then contacted me, and we were able to show that SGML and T_EX and others were predated by the APL Text Machine.

■ On-line documents in court

From 1969 to 1982, the United States Government pursued a major lawsuit against IBM regarding the unbundling of software and services^[114,115]; this 13-year antitrust lawsuit was eventually dropped in 1982. But while it was in process, it was a huge “you bet your company” undertaking, where the IBM legal team knew that the entire future of the IBM Corporation was dependent upon the evidence that they could present in court about how IBM was conducting its business.

With so much documentation that could be called up in court, they wanted to have *on-line*

access to documents in court, so that any of perhaps millions of memos and other documents could be called up instantly. These days, of course, any large case would do that, but back then, it was a brand-new concept to bring a terminal into the courtroom to call up documents pertinent to the case.

They were doing all this with APL, and in fact, were doing it on our Kingston APL time-sharing system. Chuck Norcutt and I got involved to assist with it. One of problems was that the code had been written by a summer student and it was too slow —that old story—and so of course, we said, “Okay; let’s look at the code.” *Gasps* from the legal team! You can’t look at the code — it’s *Registered IBM Confidential*, and it’s very, very sensitive, so absolutely not — *no one* can look at the code. Well then, the big challenge became, *how do you speed up code when you’re not allowed to look at it?* Is that even possible?

Surprisingly, the answer to that is *yes*. It turned out that an outgrowth of that court case was an APL application that Chuck Norcutt worked on in my department (with a little help from me), called “Pareto.” Vilfredo Pareto (1848–1923)^[116] was an Italian economist, and one of his findings was that around 80% of the result of almost anything is generally based on about 20% of its components. Applying his precepts to code, you turn on the timer and then just run your application normally; its timing analysis will show you the 20% of the code that’s taking most of the time, and then you can fix *just that portion* — the rest of it isn’t nearly as consequential. That timing facility later became the APL *Application Performance Analysis*^[117,118] tool that is now *built-in* to the APL2 interpreter.

■ **If APL is so good, where are the derivative languages?**

Morgan Stanley’s Aplus^[12] (or “A+”) was derived from other APL systems— some people call Aplus a derivative language, and some people simply consider it to be another APL dialect, just as Sharp APL and IBM APL2 differ somewhat, but both are APL dialects.

Ken Iverson’s J language^[119,120] (1990) was a direct outgrowth of APL, and it answers the question, *if you had to do it all over again, what would you change?* He couldn’t simply choose to change the APL language because of the impact to the installed user base. J was a brand-new language, to get around that issue. And now K^[121,122] (1993) is a further derivative, and Kx^[123] (1993) grew from that.

Additionally, ALGOL 68 was greatly influenced by APL, and even used the APL type ball for coding. Matlab and Mathematica are also systems which derived from APL.

Detouring for just a moment, perhaps you have seen “Zippy the Pinhead” comics in the

newspapers. Shown below is the “Symbol-Minded” strip^[124] where Zippy is trying to figure out what “J” is doing.

And now let’s talk about one more derivative of APL...

■ **Spreadsheets**

My final topic here is on spreadsheets: Where do spreadsheets come from? If you look up spreadsheets, for instance in Wikipedia,^[125] you’ll find some discussion of IBM’s *Financial Planning and Control System*^[126] from 1976. It was used in 30 countries around the world, and is considered to be an early spreadsheet. It was written in APL, but the users didn’t see APL — the underlying language was hidden from them.

The next step in this is credited to Dan Bricklin.^[127,128] He was an APL user^[129] and he reportedly admired the way that APL could display a matrix of numbers on the screen—but wouldn’t it be convenient if you could run the cursor up and just overtype a value that you wanted to change, rather

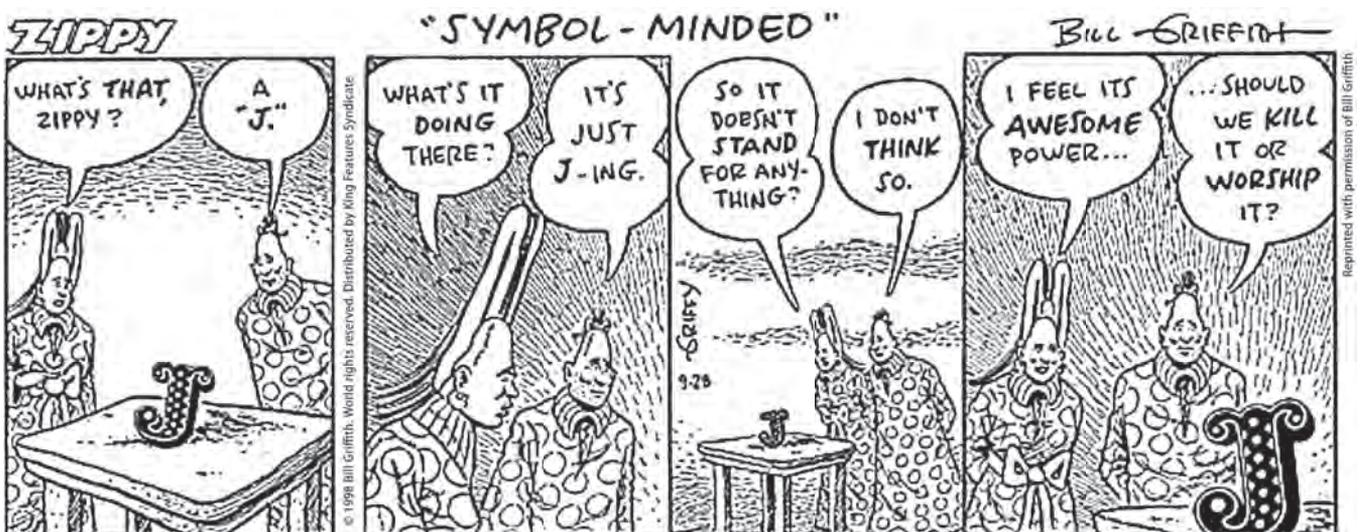


Figure: ‘Zippy the Pinhead’ contemplates J^[124]

than having to index into an array. And of course you would then want to have it recalculated automatically. So while he was a student at the Harvard Business School, he developed VisiCalc^[130,131] to solve some of what he saw as limitations in APL. This was in 1979, and that one application is widely credited with fueling the rapid growth of the personal computer industry. From that came Lotus 123 and Excel and others. He was presented with the Grace Murray Hopper Award in 1981^[22] for VisiCalc. I will point

out that spreadsheets are therefore a direct outgrowth of APL.

■ Concluding remarks

APL's mark extends far beyond just its notation. APL has had some major influences on many portions of the computer industry, and in many cases, we don't even recognize features as being related to APL anymore, but APL's influences are all around us, even in unexpected places. ■

Author: Jon McGrew, IBM (retired); Kingston, New York; phone: +1-845-338-5558, email: McGrew@TypeMatters.com

Jon McGrew has specialized in working with various flavors of the APL programming language throughout most of his career, since 1971. His focus has been on the design and development of the language itself (as opposed to the underlying implementations), and on applications. In 1981, McGrew developed one of the early widespread email systems, with service to seventeen countries; it was, of course, implemented entirely in APL.

He has spent his career working as both an APL programmer and a technical writer, and is the author of widely-used programming texts (*An Introduction to APL2*,^[132-135] and the IBM APL periodical *Jot Dot Times*^[37,111,136]). For many years, McGrew hosted international APL conferences twice each year around the world, as the Chairman of the ITL (Interdivisional Technical Liaison) Committee on the APL language.^[31] He received the IBM *President's Award* for service to the IBM APL

community. He was Team Leader for APL applications and customer support within IBM. Later, he was an APL applications writer within IBM's *Numerically Intensive Computing* group, providing support for APL on supercomputers. He then joined IBM's *APL Language Development* group under IBM's *Scientific Languages*, where he did applications programming for APL products. On the side, he volunteered for twelve years as the Production Editor for SIGAPL's *APL Quote Quad* under ACM (the Association for Computing Machinery), the programming community's major professional society.

Later, McGrew worked in the Aplus Development and Support group^[13] at Morgan Stanley Dean Witter, supporting the Aplus language^[12] and developing and teaching classes for their in-house version of APL.

McGrew is the 2001 recipient of the industry-wide Iverson Award (*The Kenneth E. Iverson Award for Outstanding Contribution to the Development and Application of APL*),^[137] presented at the international APL conference, held at Yale University.

■ Cited references, further information, and notes

- [1] *A Programming Language*, by Kenneth E. Iverson, 1962: John Wiley & Sons, Inc.; ISBN-10: 0471430145; ISBN-10: 0471430145; Computer History Museum: <http://www.softwarepreservation.org/projects/apl/Books/APROGRAMMING%20LANGUAGE/view>
- [2] *A Programming Language*, by Kenneth E. Iverson, 1962: John Wiley & Sons, Inc., ISBN:0-471430-14-5: <http://www.jsoftware.com/papers/APL.htm>
- [3] *A Programming Language*, by Kenneth E. Iverson, 1962: John Wiley & Sons, Inc.; ISBN-10: 0471430145; ISBN-10: 0471430145; Amazon: <https://www.amazon.com/Programming-Language-Kenneth-Iverson/dp/0471430145>
- [4] *IBM Systems Journal*, Volume 30, Issue 4, December 1991 (Ray Polivka, guest editor; Jon McGrew, typography and production): <http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=5387434>
- [5] *A Personal History of APL*, Michael S. Montalbano: <http://ed-thelen.org/comp-hist/APL-hist.html#PersonalHistory>
- [6] APL Design Group photo is from the Computer History Museum in California, http://www.computerhistory.org/atchm/wp-content/uploads/2012/10/iverson_team.jpg
- [7] APL Design Group photo, outtake version: <http://lathwellproductions.ca/wordpress/2010/04/21/unsung-jedi-warrior/>
- [8] ACM Conference photo is from the collection of Raymond P. Polivka
- [9] IBM 2741 terminal: https://en.wikipedia.org/wiki/IBM_2741
- [10] IBM 2741 terminal: http://www.textfiles.com/bitsavers/pdf/ibm/27xx/GA24-3415-3_2741_Data_Terminal_Aug72.pdf
- [11] IBM APL2: <http://www-03.ibm.com/software/products/en/apl2>
- [12] Morgan Stanley's Aplus language: <http://www.aplusdev.org>
- [13] Morgan Stanley's Aplus development and support team: <http://www.aplusdev.org/Develop/devTeam.html>
- [14] *I P Sharp Associates and the Telephone Monopolies*, Ian P. Sharp: <http://rogerdmoore.ca/INF/EIPSPTa.html>
- [15] Packet switching: https://en.wikipedia.org/wiki/Packet_switching
- [16] Packet-Switching History, Roger D. Moore: <http://rogerdmoore.ca/PS/>
- [17] IPSANET: <https://en.wikipedia.org/wiki/IPSANET>
- [18] IPSANET Documents, Roger D. Moore: <http://rogerdmoore.ca/INF/>
- [19] Grace Murray Hopper Award: https://en.wikipedia.org/wiki/Grace_Murray_Hopper_Award
- [20] Grace Murray Hopper Award, ACM: <http://awards.acm.org/hopper>
- [21] RADM Grace Hopper: https://en.wikipedia.org/wiki/Grace_Hopper
- [22] Grace Murray Hopper Award Recipients: https://en.wikipedia.org/wiki/Grace_Murray_Hopper_Award#Recipients
- [23] Grace Murray Hopper Award text: https://en.wikipedia.org/wiki/Richard_H._Lathwell
- [24] Grace Murray Hopper Award announcement letter for Richard Lathwell, from ACM Awards Committee Chairman, American Institute of Physics, May 31, 1973: <https://aprogramminglanguage.files.wordpress.com/2010/02/hopperdad.jpg>
- [25] *APL\360: An Interactive Approach*, John Wiley and Sons, Inc., Leonard Gilman and Allen J. Rose, 1970-01-01, ISBN-13:9780471300205, 335 pgs: <http://www.softwarepreservation.org/projects/apl/Books/GillmanAndRose>

- [26] Early time-sharing systems and APL, Eugene McDonnell, *The Socio-Technical Beginnings of APL*, APL Quote-Quad, Volume 10, Number 2, 1979-12: <http://www.jsoftware.com/papers/eem/socio1.htm#early>
- [27] *Management of APL public libraries*, Marguerite A. Boisvert, Systems Analyst, APL '79 Proceedings of the international conference on APL: part 1, pp 381–384: <http://dl.acm.org/citation.cfm?doid=800136.804491>
- [28] Self-replicating APL expressions: http://gopher.quux.org:70/Archives/usenet-a-news/NET.lang.apl/82.04.26_uwvax.343_net.lang.apl.txt
- [29] First computer virus, per BBC: <http://news.bbc.co.uk/2/hi/technology/8366703.stm>
- [30] Elk Cloner virus: https://en.wikipedia.org/wiki/Elk_Cloner
- [31] Proceedings of the IBM ITL (Interdivisional Technical Liaison) Committee on the APL language, Computer History Museum, <http://www.computerhistory.org/collections/catalog/102734222>
- [32] *Internet in its infancy*, ActewAGL Retail, ABN 46 221 314841, <https://web.archive.org/web/20110227151622/http://www.actewagl.com.au/Education/communications/Internet/historyOfTheInternet/InternetOnItsInfancy.aspx>
- [33] Mailbox, *The STSC Story: It's About Time*: <https://www.youtube.com/watch?v=BSkxr6rQU0Y>
- [34] 666 BOX, *APL Quotations and Anecdotes*, Roger Hui: <http://www.jsoftware.com/papers/APLQA.htm#666box>
- [35] 666 BOX was the I. P. Sharp mail system: https://en.wikipedia.org/wiki/I._P._Sharp_Associates
- [36] *Will You Love Electronic Mail or Hate It?* Computer Decisions, Volume 11, Hayden Publishing Company, December 1979, pg 47
- [37] *The APL Jot Dot Times*: An IBM in-house newsletter created by Jon McGrew; “Introducing the APL Mailbox,” Summer 1981, 64 pages
- [38] IBM VNET: https://en.wikipedia.org/wiki/IBM_VNET
- [39] *Standardizing Network Mail Headers*, IETF Network Working Group, RFC 561, September 1973: <https://tools.ietf.org/html/rfc561>
- [40] *Simple Mail Transfer Protocol (SMTP)*, IETF Network Working Group, RFC 821, August 1982: <https://tools.ietf.org/html/rfc821>
- [41] Dyalog APL: <https://www.dyalog.com/>
- [42] *There's DNA Everywhere*, SIGPLAN Chapter on Array Programming Languages, Charles Brenner: <http://www.sigapl.org/CharlesBrennerDNATalkIntro.php>
- [43] Charles Brenner, DNA Identification Technology and APL: <http://dna-view.com/DNAtchID.htm>
- [44] *There's DNA Everywhere— an Opportunity for APL*, Charles Brenner: <http://video.dyalog.com/Dyalog14/?v=oXIP3r6PzeE>
- [45] Charles Brenner (mathematician): [https://en.wikipedia.org/wiki/Charles_Brenner_\(mathematician\)](https://en.wikipedia.org/wiki/Charles_Brenner_(mathematician))
- [46] Charles Brenner, *A Conversation With Charles Brenner; A Math Sleuth Whose Secret Weapon Is Statistics*, NY Times, Science section, By Claudia Dreifus, Aug. 8, 2000: <http://www.nytimes.com/2000/08/08/science/conversatipon-with-charles-brenner-math-sleuth-whose-secret-weapon-statistics.html>
- [47] *Language as an Intellectual Tool: From Hieroglyphics to APL*, Donald B. McIntyre, IBM Systems Journal, Volume 30, Issue 4, December 1991, pp 554–581: <http://ieeexplore.ieee.org/document/5387447/?arnumber=5387447>
- [48] *Language as an Intellectual Tool: From Hieroglyphics to APL*, Donald B. McIntyre (1991), IBM Systems Journal. 30 (4): 554–581: <http://domino.research.ibm.com/tchjr/journalindex.nsf/e90fc5d047e64ebf85256bc80066919c/9c834f5a16efa82085256bfa00685c72!OpenDocument>
- [49] National Language Translation: APL2 Programming: Language Reference, SH21-1061-01, February 1994, pg 314: <http://publibfp.boulder.ibm.com/epubs/pdf/h2110611.pdf>

- [50] Apple I computer: <http://www.computerhistory.org/revolution/personal-computers/17/312/2312>
- [51] Apple I computer: https://en.wikipedia.org/wiki/Apple_I
- [52] Homebrew Computer Club: https://en.wikipedia.org/wiki/Homebrew_Computer_Club
- [53] Homebrew Computer Club: <http://www.computerhistory.org/revolution/personal-computers/17/312>
- [54] Homebrew Computer Club: <http://www.oldcomputers.net/applei.html>
- [55] *Steve Wozniak Debunks One of Apple's Biggest Myths*: <https://www.youtube.com/watch?v=pJif4i9NRdI>
- [56] Apple II computer: <http://www.computerhistory.org/revolution/personal-computers/17/300>
- [57] Apple II computer: https://en.wikipedia.org/wiki/Apple_II
- [58] Apple II computer: https://en.wikipedia.org/wiki/Apple_II_series
- [59] Apple II computer: <http://oldcomputers.net/appleii.html>
- [60] SCAMP: http://www-03.ibm.com/ibm/history/exhibits/pc/pc_1.html
- [61] SCAMP presentation to the Smithsonian Institute, Paul Friedl, IBM Corporation *SCAMP - The 1st IBM personal computer; the missing link in the PC's past!*: https://www.youtube.com/watch?v=L_BWL7vCaa0
- [62] *World's first PC*: PC Magazine, Vol. 2, No. 6, November 1983, Ziff-Davis Publishing, "SCAMP: The Missing Link in the PC's Past?", pp 191–197: https://books.google.com/books?id=q8fwTt09_MEC&pg=PA196&lpg=PA196&dq=PC+Magazine,+November+1983+scamp
- [63] IBM 5100: http://www-03.ibm.com/ibm/history/exhibits/pc/pc_2.html
- [64] IBM 5100: https://en.wikipedia.org/wiki/IBM_5100
- [65] IBM 5100 ad: <http://bluefaqs.com/2009/09/35-vintage-tech-ads/>
- [66] John Titor and the 5100: https://en.wikipedia.org/wiki/John_Titor
- [67] IBM 1130 computer: https://www-03.ibm.com/ibm/history/exhibits/1130/1130_intro.html
- [68] IBM 1130 computer: https://en.wikipedia.org/wiki/IBM_1130
- [69] *How We Got to APL 1130*, Larry Breed, Vector (British APL Association), 22 (3), August 2006, ISSN 0955-1433: <http://vector.org.uk/art10001190>
- [70] MCM/70, *How Toronto invented the PC, then forgot about it*: <http://spacing.ca/toronto/2015/04/15/toronto-invented-pc-forgot/>
- [71] *The MCM/70 Microcomputer*, by Zbigniew Stachniak (developer): http://www.xnumber.com/xnumber/MCM_70_microcomputer.htm
- [72] *Inventing the PC: The MCM/70 Story*, book, by Zbigniew Stachniak (developer): <https://www.amazon.com/Inventing-PC-MCM-70-Story/dp/0773538526>
- [73] *Inventing the PC: The MCM/70 Story*, review, David C. Brock: <https://muse.jhu.edu/article/476817>
- [74] *The Making of the MCM/70 Microcomputer*, by Zbigniew Stachniak, IEEE Annals of the History of Computing, April–June 2003, pp. 62–75: <https://ia801300.us.archive.org/10/items/MCM01203059/MCM-01203059.pdf>
- [75] MCM/70, *Core 4.1*, Computer History Museum, September 2003, pp 6–12: <http://s3data.computerhistory.org/core/core-2003.pdf>
- [76] *Introduction of the MCM/70, the First Truly Portable Computer & the First Truly Usable Microcomputer System*, Jeremy Norman, History of Information: <http://www.historyofinformation.com/expanded.php?id=4806>
- [77] MCM/70 used on a plane, Zbigniew Stachniak, from *APL Quotations and Anecdotes* by Roger Hui: <http://www.jsoftware.com/papers/APLQA.htm#MCM1973>
- [78] *APL Quotations and Anecdotes*, Jsoftware, compiled and edited by Roger Hui: <http://www.jsoftware.com/papers/APLQA.htm>

- [79] *MCM/70 и другие: из истории персональных компьютеров* (“MCM/70 and others: from the history of personal computers”), 08.09.2013: <http://itc.ua/articles/mcm-70-i-drugie-iz-istorii-personalnyih-kompyuterov/>
- [80] APL in Russia, *Russia: a future great APL power?*, Erkki Juvonen, ACM SIGAPL APL Quote Quad - Russian focus issue, Volume 22 Issue 2, Dec. 1991, pp 1–2: <http://dl.acm.org/citation.cfm?doid=130647.130649&CFID=797566137&CFTOKEN=80570900>
- [81] APL in Russia, *Nuclear power plant diagnostics in APL*, Alexander O. Skomorokhov, Institute of Physics and Power Engineering, 1 Bondarenko Square, Obninsk, Kaluga Region 249020, USSR, APL ’91 Proceedings of the international conference on APL ’91, pp 289–300: <http://dl.acm.org/citation.cfm?id=114087&CFID=797566137&CFTOKEN=80570900>
- [82] APL in Russia, *The SovAPL award for excellence in APL*, Alexander Skomorokhov, Chairman of SovAPL, The Russian Chapter of ACM SIGAPL, ACM SIGAPL APL Quote Quad, Volume 31 Issue 1, 09/01/2000, pp 29–30: <http://dl.acm.org/citation.cfm?id=570511&CFID=797566137&CFTOKEN=80570900>
- [83] Altair 8800, Popular Electronics, Jan 2, 1975, pp 33–38: <http://www.americanradiohistory.com/Archive-Poptronics/70s/1975/Poptronics-1975-01.pdf>
- [84] Altair 8800, Setup and Users Manual, vintage computer information, published by Briel Computers, July 2010: <http://www.hackersinformation.com/uploads/1/9/1/6/19169525/micromanual.pdf>
- [85] Ampere WS-1: https://fi.wikipedia.org/wiki/Ampere_WS-1 (Finnish)
- [86] *The Amazing Ampere WS-1* (1985), Norbert Landsteiner: <https://plus.google.com/+NorbertLandsteiner1/posts/VJYZrHZg7Zy>
- [87] Ampere WS-1, Classic Tech, Vintage computers and related technology, Ampere Inc. (Tokyo, Japan): <https://classictech.wordpress.com/computer-companies/ampere-inc-tokyo-japan/>
- [88] Ampere WS-1: <http://www.old-computers.com/museum/computer.asp?st=1&c=66>
- [89] Ampere WS-1, *The Computer Chronicles— Japanese PCs (1984)*: <https://www.youtube.com/watch?v=rbh1XP4kCT4>
- [90] *The Guy Who Designed the Datsun 240Z Also Designed This Fascinating Laptop*, Jalopnik: <http://jalopnik.com/5929191/the-guy-who-designed-the-datsun-240z-also-designed-this-fascinating-laptop>
- [91] *Datsun 240Z Exterior Designer “Kumeo Tamura”*: <https://www.youtube.com/watch?v=ju7RfHhTI1c>
- [92] Bill Gates meeting at IPSA, *APL Quotations and Anecdotes*: http://www.jssoftware.com/papers/APLQA.htm#Bill_Gates
- [93] *An Open Letter to Hobbyists*, Homebrew Computer Club Newsletter, Vol 2, Issue 1, February 3, 1976, pg 2, by Bill Gates; DigiBarn Computer Museum: http://www.digibarn.com/collections/newsletters/homebrew/V2_01/homebrew_V2_01_p2.jpg
- [94] *APL: Good for the Brain*, article by Bill Gates, Electronics Today International Magazine (Canada), Vol. 3 No. 3, March 1979, by Steve Braidwood (editor), March 1979
- [95] Hewlett-Packard HP-35 scientific calculator: <https://en.wikipedia.org/wiki/HP-35>
- [96] Hewlett-Packard HP-35 scientific calculator designed with APL\1130: Personal email discussion with Richard Lathwell, 2017-01-04
- [97] IBM Vector Facility: http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_FS9000.html
- [98] *IBM Vector Facility: Large Systems Technical Support*, IBM 3090 Processor Complex: Planning and Installation Reference, GG66-3090-01, November 1987, http://chiclassiccomp.org/docs/content/computing/IBM/Mainframe/Hardware/System/GG66-3090-01_3090ProcComplexPlanningInstallationRef_Nov87.pdf

- [99] IBM Vector Facility: https://en.wikipedia.org/wiki/IBM_3090#Vector_facility
- [100] Prof. Dr. Meinzer, DKFZ: Descriptions taken from videotape liner notes created by Jon McGrew; video program created by John M. Mizel, editing by Jon McGrew and Mike Van Der Meulen; IBM Numerically-Intensive Computing, February 1990. The video was created for the IBM NYC exhibit entitled, “100 Years of Computing.”
- [101] Judson Rosebush, Wikipedia: https://en.wikipedia.org/wiki/Judson_Rosebush
- [102] Digital Effects, Inc., Wikipedia: [https://en.wikipedia.org/wiki/Digital_Effects_\(studio\)](https://en.wikipedia.org/wiki/Digital_Effects_(studio))
- [103] Digital Effects, Inc., from comments made at the presentation by Judson Rosebush at the I. P. Sharp Conference, Toronto, 1982
- [104] Xanadu movie trailer, 1980: <https://www.youtube.com/watch?v=WNcUv1q2JAs&t=23s>
- [105] Tron movie, 1982, *Film Opening at Flynn’s Arcade*: <https://www.youtube.com/watch?v=P7LclSGFOFg&t=12s>
- [106] Tron movie, 1982, *The Making of Tron*, Director’s comments: <https://www.youtube.com/watch?v=pr2LvJUI6ZY&t=1s>
- [107] APL Text Machine history: Personal email discussion with Richard Lathwell, 2017-01-04
- [108] Selectric typing element: http://www-03.ibm.com/ibm/history/exhibits/vintage/vintage_4506VV2122.html
- [109] Selectric typing element: https://en.wikipedia.org/wiki/IBM_Selectric_typewriter
- [110] Selectric typing element: <http://www.computerhistory.org/collections/catalog/102662795>
- [111] *The APL Jot Dot Times*: An IBM in-house newsletter created by Jon McGrew; “Special Reference Issue,” Fall 1980, 98 pages, Computer History Museum: <http://www.softwarepreservation.org/projects/apl/Papers/APLJotDotTimes>
- [112] Pre-APL Text Machine manual, *APL\360 Language and Time Sharing System*, 1968: http://bitsavers.trailing-edge.com/pdf/ibm/apl/APL_360_Features.pdf
- [113] Donald Knuth usage of APL: Personal email discussion with Richard Lathwell, 2017-01-04
- [114] IBM antitrust lawsuit; Wikipedia, History of IBM: https://en.wikipedia.org/wiki/History_of_IBM#1969:_Antitrust.2C_the_Unbundling_of_software_and_services
- [115] IBM antitrust lawsuit; New York Times, February 15, 1981, U.S. vs. I.B.M.: <http://www.nytimes.com/1981/02/15/business/us-vsibm.html>
- [116] Vilfredo Pareto: https://en.wikipedia.org/wiki/Vilfredo_Pareto
- [117] APL2 timing facility, *TIME— Application Performance Analysis*: APL2 User’s Guide, IBM, SC18-7021-23, 1994/2017, pp 287–288; <http://publibfp.boulder.ibm.com/epubs/pdf/c187021n.pdf>
- [118] APL2 timing facility, *TIME Workspace*: APL2 User’s Guide, IBM, SC18-7021-22, 1994/2017, pp 547–549; <http://publibfp.boulder.ibm.com/epubs/pdf/c187021n.pdf>
- [119] J language: <http://www.jsoftware.com/>
- [120] J language: [https://en.wikipedia.org/wiki/J_\(programming_language\)](https://en.wikipedia.org/wiki/J_(programming_language))
- [121] K language, Vector, Vol. 10 No. 1, July 1993, pp 74–79: <http://archive.vector.org.uk/art10010830>
- [122] K language, Wikipedia: [https://en.wikipedia.org/wiki/K_\(programming_language\)](https://en.wikipedia.org/wiki/K_(programming_language))
- [123] Kx Systems, 1993: <https://kx.com/>
- [124] *Zippy the Pinhead*, © 1998, Bill Griffith; published on September 28, 1998; Used here through the permission of the artist: https://en.wikipedia.org/wiki/Zippy_the_Pinhead
- [125] Spreadsheet, Wikipedia: <https://en.wikipedia.org/wiki/Spreadsheet>
- [126] Spreadsheet; *IBM Financial Planning and Control System (FPCS)*, Wikipedia: https://en.wikipedia.org/wiki/Spreadsheet#IBM_Financial_Planning_and_Control_System

- [127] Dan Bricklin's website: <http://bricklin.com>
- [128] Dan Bricklin, Wikipedia: https://en.wikipedia.org/wiki/Dan_Bricklin
- [129] Dan Bricklin on APL; *Bricklin on Technology*, Dan Bricklin, John Wiley and Sons, Inc., 2009, ISBN-13: 978-0470402375, pp 370, 401–404, 423; <http://www.barnesandnoble.com/w/bricklin-on-technology-daniel-bricklin/1102658682?ean=9780470402375>
- [130] VisiCalc background, from Dan Bricklin's website: <http://bricklin.com/visicalc.htm>
- [131] VisiCalc: <https://en.wikipedia.org/wiki/VisiCalc>
- [132] *An Introduction to APL2* (Installed User Program), IBM, Jon McGrew, June 1982, SB21-3039-0
- [133] *An Introduction to APL2*, APL2 Version 1 Release 2 (Program Product), IBM, Jon McGrew, December 1985, SH20-9229-1, Computer History Museum: <http://www.computerhistory.org/collections/catalog/102679862>
- [134] *An Introduction to APL2*, APL2 Version 1 Release 2 (Program Product), IBM, Jon McGrew, December 1985, SH20-9229-1, Bitsavers: https://archive.org/details/bitsavers_ibmaplSH20toAPL2Dec85_12392164
- [135] *An Introduction to APL2*, APL2 Version 2 Release 1 (Program Product), IBM, Jon McGrew, March 1992, SH21-1073-00: <http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=SH21-1073-00>
- [136] *The APL Jot Dot Times*: An IBM in-house newsletter created by Jon McGrew; "Special Security Issue," Spring 1985, 234 pages, Computer History Museum: <http://www.softwarepreservation.org/projects/apl/Brochures/APLJotDotTimes>
- [137] Iverson Award: https://en.wikipedia.org/wiki/Iverson_Award

~

James A. Brown* and Peter Schade**

The Evolution of Computing

On the occasion of the 50th anniversary of the APL workspace 1 CLEANSPACE German Guide Share Europe Working Group, APL Germany and IBM Germany
November 27-29, 2016 IBM Böblingen, Germany

This paper is extracted from the HTML presentation made at the conference. Without the discussion that accompanied the presentation, not everything is easily understood. However, some text is added below for clarification.

Motivation

1. Processors

- a. Lots of processors on a chip
 - i. My laptop has 8 cores
 - ii. Intel 48-Core "Single-Chip Cloud Computer"
 - iii. Adapteva Epiphany architecture allows 2048 processors on a chip
- b. All share the same main memory
- c. Memory contention limits performance
 - i. levels of cache memory - store information the CPU is most likely to need next
 - ii. sophisticated algorithms - it gets complicated

2. Main Memory

- a. Random-access memory
 - i. Can read or write data in the same time no matter physical location
 - ii. Usually volatile - information lost when power removed
 - iii. SRAM - Static random-access memory (Faster but needs more power)
 - iv. DRAM - dynamic random-access memory

3. Attached Memory

- a. Spinning Disk Drives
 - i. Formatted with Tracks, Cylinders, Sectors
- b. Solid State Disk Drives
 - i. Simulates spinning drive
 - ii. Perfect compatibility
 - iii. zero seek time
 - iv. Very fast
 - v. wastes the potential

* NestedComputing Corp; SmartArrays, Inc.

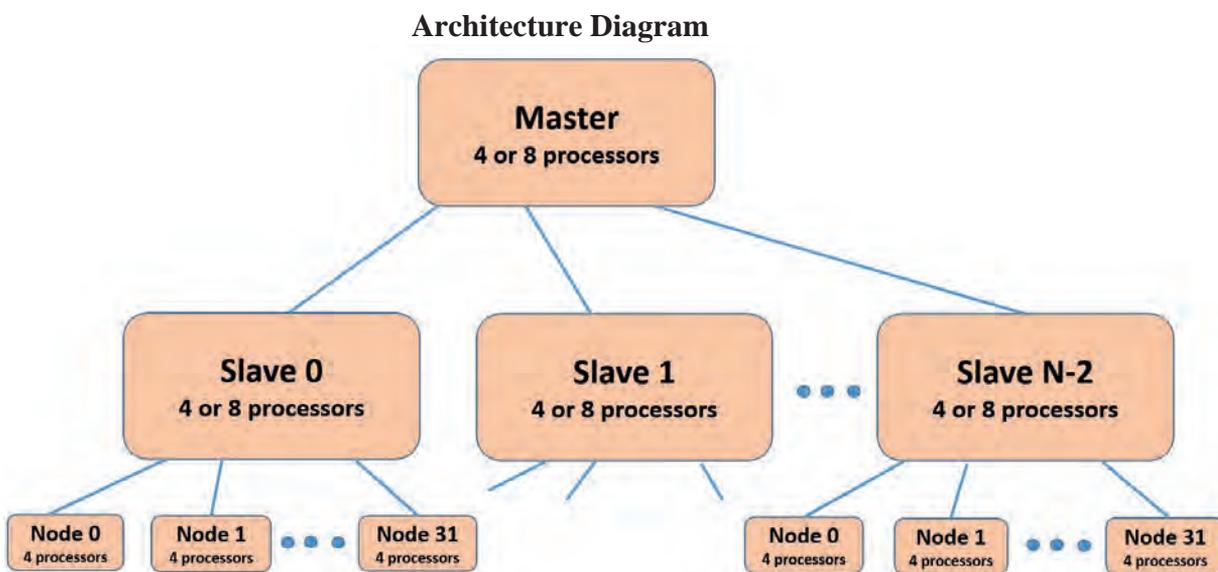
** International Microsystems

- c. NVMe Disk Drive
 - i. NVMe - Non-Volatile Memory Express
 - ii. Does not simulate spinning drive
 - iii. Requires new OS support (Linux has it)
 - iv. Significantly faster data transfer
 - v. Requires fewer CPU cycles

You might wonder why the detailed discussion of NVMe disks. It's because it is the current state of the art in attached memory and the possible next generation of this technology is part of the future machine described below.

What we're doing: DAC - Disjoint Array Computer

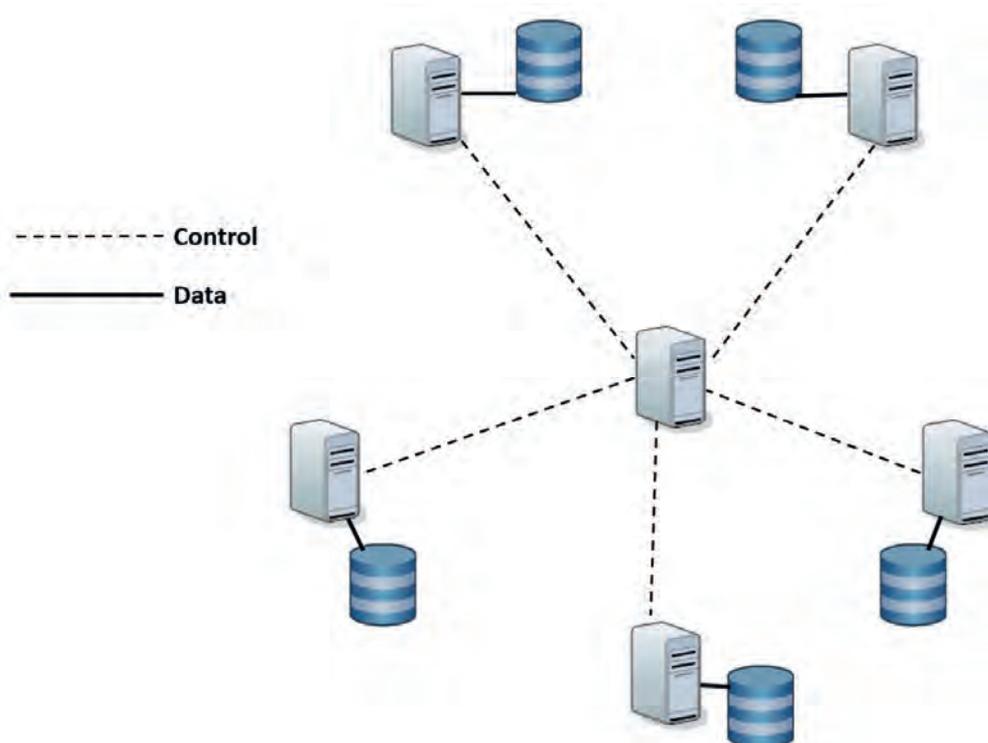
- 1. A three level machine: Machine Architecture
 - a. Three Levels of Processing
 - i. Level 1 - User and External Network Interface
 - ii. Level 2 - Program Analysis and Distribution
 - iii. Level 3 - Arrays of Processing Nodes



- b. Large Distributed Memory – 100's of Terabytes Possible
 - c. Node memories independent at compute time
 - d. The DAC Computer
 - i. Packaged as a single computer
 - ii. Cloud in a box
 - iii. Closely coupled distributed storage
 - iv. Array data is local or remote as needed
- 2. DAC Memory Model
 - a. RAM
 - i. Each Master, Slave, and node has private RAM
 - b. Attached Memory
 - i. Each Master, Slave, and node can have private attached memory
 - ii. Slave, and Node can share attached memory (Called DAC Disks)

c. DAC Disks

- i. Hard attached to Slave or Node
 - 1. Instantaneous (Hardware switch)
 - 2. Move mass data between Slave and Node with NO Network activity
 - 3. No TCP/IP to transmit a file
 - 4. No transmission at all
- ii. Disjoint Memory
 - 1. Sometimes attached memory is local (on the slave)
 - 2. Sometimes attached memory is remote (on the nodes)
 - 3. Unlike Distributed memory
- iii. Disjoint Array
 - 1. Parts of array stored in multiple disjoint memories
 - 2. Sometimes all of array is local
 - 3. Sometimes array is distributed
- iv. DAC Disk Connections in following diagrams
 - 1. Control Line: - - - - -
 - a. Slow Speed Hardware Connection
 - b. Ethernet over USB (a private internet connection)
 - c. For commands, return codes, messages
 - 2. Data Line: _____
 - a. High Speed Hardware Connection
 - b. For high volume data
- v. DAC Memory Model: View 2 - DAC Disks attached to Nodes



3. Disjoint Array

- a. "Disjoint" From Synonym.com
 - i. Separated
 - ii. Divided
 - iii. Disassembled
- b. A Disjoint array is:

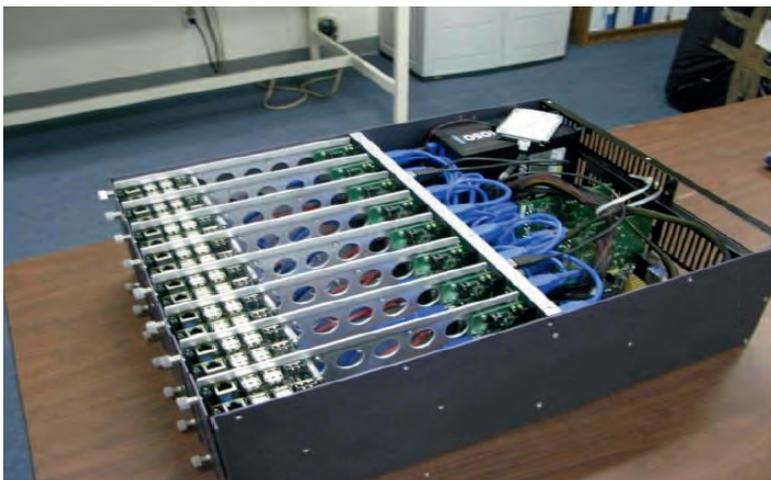
An Array that leverages the DAC Memory model and is sometimes locally stored and sometimes remotely stored without the movement of data

The new capability of the DAC machine is the ability to do parallel processing with no memory contention because the attached memories are hardware movable between slaves and nodes.

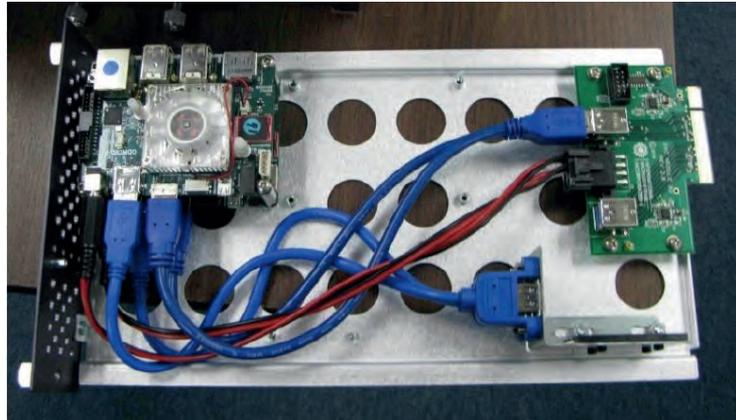
4. Three Prototypes of DAC Machine have been built
 - a. DAC Prototype 1
 - i. Master / Slave: M5208 Intel 64-bit USB Duplicator
 - ii. 8 Nodes: ODROID-XU ARM 32-bit
 - iii. Used for Software Development



- b. DAC Prototype 2
 - i. Master / Slave: Custom Packaging Intel 64-bit
 - ii. 16 Nodes: ODROID-XU ARM 32-bit
 - iii. In a single box
 - iv. Used for Customer/Investor demos



One Node



c. DAC Prototype 3

- i. Master / Slave: Mini-ITX J1900 Intel 64-bit
- ii. 4 Nodes: Mini-ITX J1900 Intel 64-bit



One Node



5. Programming

a. DAC API: The commands

- active: Display a list of active nodes
- distribute: Copy unique set of files to each node
- exec2: Execute an OS command on a slave (level 2)
- exec3: Execute an OS command on a node (level 3)
- exec3r: Execute an OS command on a node (level 3) and don't wait for a result
- exec3z: Wait for then fetch exec3r result
- exit: Shut down client
- executing: List nodes that are executing an OS command
- filecopy: Copy a set of files to each node
- filedelete: Delete files on each node
- filelist: List files that exist on each node
- filelistlocal: List files that exist on slave
- mount: mount Disjoint memory on node (and remove from slave)
- umount: mount Disjoint memory on slave (and remove from node)
- restart: Shut down node server and restart with new code level
- status: Information about Disjoint memory and where it's mounted
- time: Give elapsed time since you last asked

b. Programming languages supported

- i. C
- ii. C++
- iii. C#
- iv. SmartArrays
- v. Dyalog APL

c. High Level Interfaces

i. DAC Interactive Console: Used for debugging and learning

```
Input: mount
000: OK
003: OK
001: OK
002: OK
Input: status
002: USB: localhost Device: localstore mounted on (simulated) node at
/AOSLocal 7 files 353M bytes used 11063M free space 3010M RAM
000: USB: localhost Device: localstore mounted on (simulated) node at
/AOSLocal 7 files 265M bytes used 10442M free space 3010M RAM
003: USB: localhost Device: localstore mounted on (simulated) node at
/AOSLocal 7 files 353M bytes used 10983M free space 3010M RAM
001: USB: localhost Device: localstore mounted on (simulated) node at
/AOSLocal 7 files 353M bytes used 11065M free space 3010M RAM

Input: exec3 sleep %%%
000: OK
001: OK
002: OK
003: OK
```

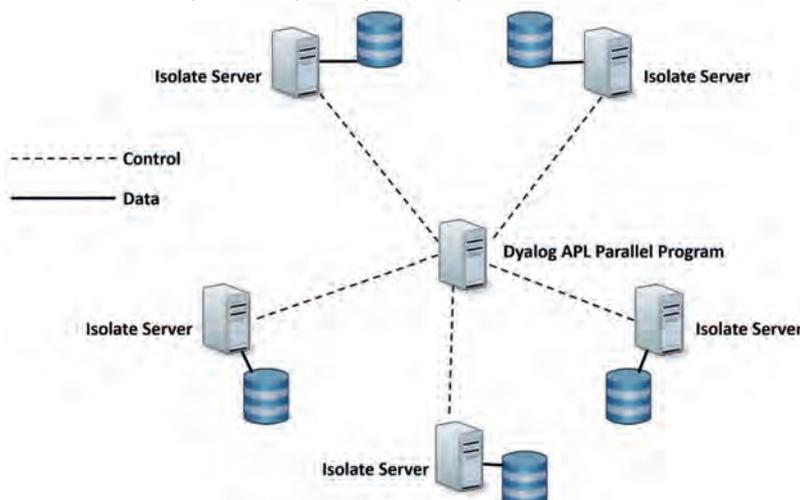
ii. DAC Controller: An Interactive Dialog

```

DAC Controller
File
Clear Close
Waiting for input ...
--> 0 filelist
00: Node directory: /AOSLocal
00: 16384 2014 Mar 27 20:27 dtdemo
00: 0 2014 Feb 24 08:42 --USB 0.txt
00: 832486 2014 Mar 27 20:25 dtdemo.jar
00: 67428 2014 Apr 04 16:23 HelloTest
00: OK
-->
--> mount
000-003: OK
--> status
000: USB: localhost Device: localstore mounted on (simulated) node at /AOSLocal 7 files
001: USB: localhost Device: localstore mounted on (simulated) node at /AOSLocal 7 files
002: USB: localhost Device: localstore mounted on (simulated) node at /AOSLocal 7 files
003: USB: localhost Device: localstore mounted on (simulated) node at /AOSLocal 7 files

```

d. Parallel Programming using Dyalog APL isolates



What we're doing: FASDAC- Flat Address Space Disjoint Array Computer

Note that the information about FASDAC is considered the intellectual property of International Microsystems, NestedComputing and SmartArrays. Copyright © 2016

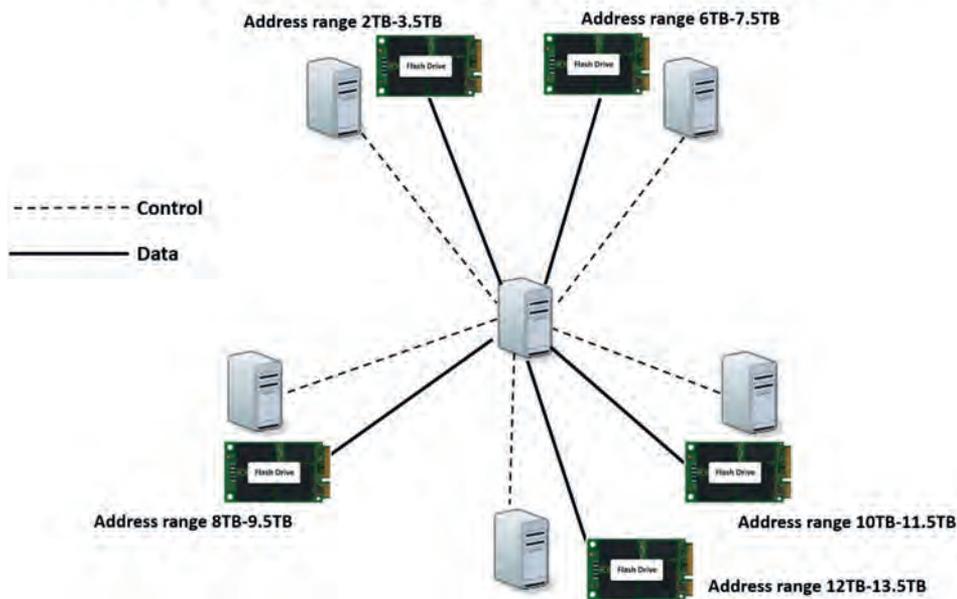
1. Flat Address Space
 - a. As opposed to segmented memory
 - b. Traditionally contiguous - zero to-memory size-1
 - c. FASDAC allows multiple contiguous address ranges
 - d. Address translation still allowed (assignment of real memory to virtual memory)
 - e. RAM - one kind of memory for immediate and long term storage
2. NVMe revisited
 - a. Significantly faster data transfer
 - b. BUT IT'S STILL A DISK
3. NVMEF - Flat NVME
 - a. Does not simulate a spinning disk
 - b. When attached, a new continuous RAM address range is visible

- c. "Plug and Play" RAM
- d. All data is always memory resident

Flat NVMe is the extension of current NVMe to allow the flash memory to be treated as RAM. Initially, this "Plug and Play" RAM would be used for memory mapped data and not for program execution but this restriction could be lifted in a next generation.

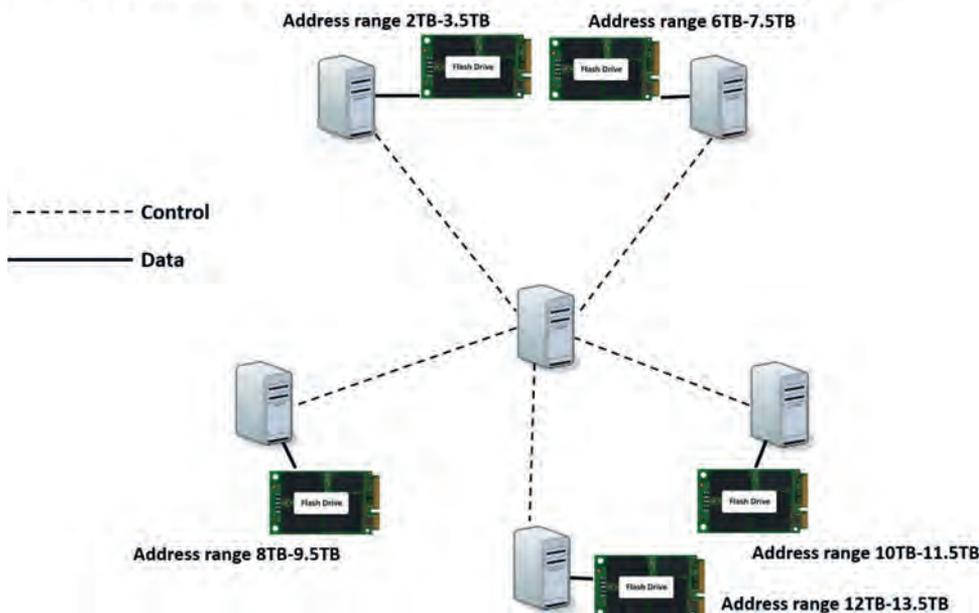
- 4. Two Views of FASDAC Memory Model
 - a. View 1 - DAC Disks attached to Slave

Slave has it's own RAM address space + five attached address spaces



- b. View 2: FASDAC Disks attached to Nodes

Each node has it's own RAM address space + an attached address space



5. Typed Memory:
 - a. Memory is "Typed"
 - i. not just a stream of bytes
 - ii. Array Oriented
 - b. Some memory is strongly typed
 - i. Type RAM - a stream of bytes for support of legacy software and program execution
 - ii. The machine's native memory is type RAM
 - iii. Type FILE - memory that contains the contents of a file
 - iv. Type OSxx - set of memory items formatted for Operating System control
 - c. Some memory is weakly typed
 - i. Type NUMERIC - a stream of numbers without respect to storage format
 - ii. Type CHAR - a stream of characters without respect to storage format (UCS 1,2,4)
 - iii. Machine supports conversion
 - d. Some memory is recursively typed
 - i. Type NESTED - memory that contains other memory entries
 - e. Attached DAC drive
 - i. Is recursively typed
 - ii. Nested structure matches directory structure
 - f. Database table
 - i. List of columns
 - ii. Each column in strongly typed memory
 - iii. Trivial memory map for analytics
 - iv. Overallocation for transactions

6. Conclusion: FASDAC is An Array Machine
 - a. Typed memory is vector oriented
 - b. Higher rank arrays realized with programming



DAC is Evolutionary



FASDAC is Revolutionary

Therefore, an operating system that is aware of typed vectors in memory running on a FASDAC machine is:

**An Array Operating System
running on an Array Machine**

Michael Baas

Bericht von Dyalog 2016

Das diesjährige Dyalog-Benutzertreffen fand vom 9-13. Oktober in Glasgow statt. Der Chronist als einer von zwei deutschen Teilnehmern möchte mit diesem Artikel gerne informieren, was da so alles geboten wurde – damit vielleicht noch weitere APLer motiviert werden und wir im nächsten Jahr schon zu dritt Skat spielen können?

Statistik

Insgesamt waren 121 Teilnehmer aus 15 Ländern (33 TN aus Großbritannien, 23 aus USA, 22 aus Italien, etlichen anderen Ländern und als Schlusslicht jeweils 2 aus Deutschland, der Schweiz, den Niederlanden und der Ukraine, jeweils 1 TN aus Süd-Afrika und Belgien). Das reine Konferenzprogramm bestand aus 25 Vorträgen an 2 Tagen, dazu kamen 1,5 Tage gemeinsamer Veranstaltung mit der British APL Association zur Feier des 50-jährigen Geburtstags. Eingerahmt war das Programm von 12 Workshops vor und nach der Konferenz.

Conference Hotel". Sowohl kulinarisch wie auch technisch wurden wir nicht enttäuscht, nur die Sache mit dem Flughafentransfer muss das Team noch ein bisschen üben ;-). Mir blieb leider keine Zeit, um neben dem Programm auch mal die Umgebung zu erkunden – etwas bedauerlich, weil das Wetter für Oktober in Schottland wirklich ausgesprochen gut war: nicht nur, dass es nicht regnete, sogar die Sonne war häufiger zu sehen! Im Rahmen des Programms wurden dann auch die lokalen Besonderheiten gewürdigt: am Abend des ersten Tages gab es eine Verkostung von Whisky und Schokolade, die eine nahegelegene Brennerei

9.10. So	10.10 Mo	11.10 Di	12.10 Mi	13.10 Do
Workshops	Dyalog	Dyalog	BAA & Dyalog	BAA & Dyalog Workshops

Insgesamt 42 Vorträge und 12 Workshops an 5 Tagen – das Programm war ziemlich "sportlich", für 2017 wünsche ich mir mehr Zeit.

Ort

Veranstaltungsort in Glasgow war das etwas außerhalb gelegene "Golden Jubilee

durchführte. Gerne hätte der Autor dieser Zeilen die schottische Whiskey-Kultur ausführlicher erforscht, doch nach 2 Gläsern war schon Schluss des Programms. Zum Festessen anlässlich der Geburtstagsfeier wurden wir dann stilecht von einem Duddel-Spieler begrüßt und dann gab es natürlich auch den berühmten "Haggis" als Teil des Menüs – stilecht zelebriert durch



einen Ureinwohner, der messerschwingend die Ode an den Haggis rezitierte.

Workshops

Die angebotenen Workshops (jeweils 4h) deckten ein breites Themenspektrum ab: im 2-teiligen "CookBook"-Workshop ging es um ein "Kochbuch" mit Empfehlungen bewährter Rezepte für das Entwickeln von Dyalog-Anwendungen; "Threading and Synchronization" zeigte Wege auf, sicher mit Threads umzugehen; bei "Web Application Development" wurde in zwei Sessions die Web-Portierung einer "alten" Desktop-Anwendung erörtert; "Compiler and Performance Features" zeigte Wege auf, die Laufzeiten zu verbessern; bei "Artificial Neural Networks in APL" ging es um die Entwicklung neuronaler Netze mit APL; im Workshop "Taming Statistics with TamStat" konnten die Teilnehmer die TamStat-Software kennenlernen, mit welcher Autor und Referent Stephen Mansour seinen Studenten den Zugang zur Statistik erleichtern möchte; Roger Hui zeigte in "A Tour (de Force) of APL in 16 expressions", wie APL die 5 wesentlichen Eigenschaften einer Notation (nach K. Iverson) auf unübertroffene Weise erfüllt; bei "Version 15.0 in Depth"

beschäftigte man sich eingehend mit den neuen Möglichkeiten von Version 15; im Workshop "Compiling ANN (Artificial Neural Networks) and other APL Code" wurde gezeigt, wie die Grafikprozessoren heutiger Computer aus Dyalog heraus genutzt werden können, um performance-kritischen, parallelisierbaren Code dort auszuführen und bei "Data Visualisation" wurde schließlich ein Überblick geboten über die vielfältigen Möglichkeiten zur Erstellung von Grafiken mit Dyalog APL. Ich schätze solche Workshops sehr, denn man hat dort die Möglichkeit, bestimmte Themen doch mal ausführlicher zu erkunden, als das im Rahmen einer Präsentation möglich ist. Für mich eine Art "Intensiv-Training", die das normale Tagungsprogramm sehr gut ergänzt.

Vorträge

Die Palette der Vorträge war zu groß, um jeden auch nur mit einer Zusammenfassung zu würdigen. Ich möchte stattdessen einfach ein paar wenige Vorträge herausgreifen, die mich besonders interessiert haben.

Nick Nickolov & Morten Kromberg: Open Front Ends

Ein kleiner Einblick in Dinge, die gerade in Entwicklung sind: so wird etwa an einem Eclipse-Plugin gearbeitet, mit dem APL-Sourcecode verwaltet werden kann. Ergänzt wird das Ganze durch einen unter Eclipse nutzbaren Debugger. RIDE 4.0, die Entwicklungsumgebung für "entfernte Entwicklung", soll die Default-IDS unter Linux und macOS werden. damit soll dann auch

eine "zero footprint"-Entwicklung möglich werden, bei der man sich über dem Webbrowser mit einer laufenden APL-Session verbinden und diese debuggen kann. Spannende Zukunftsaussichten!

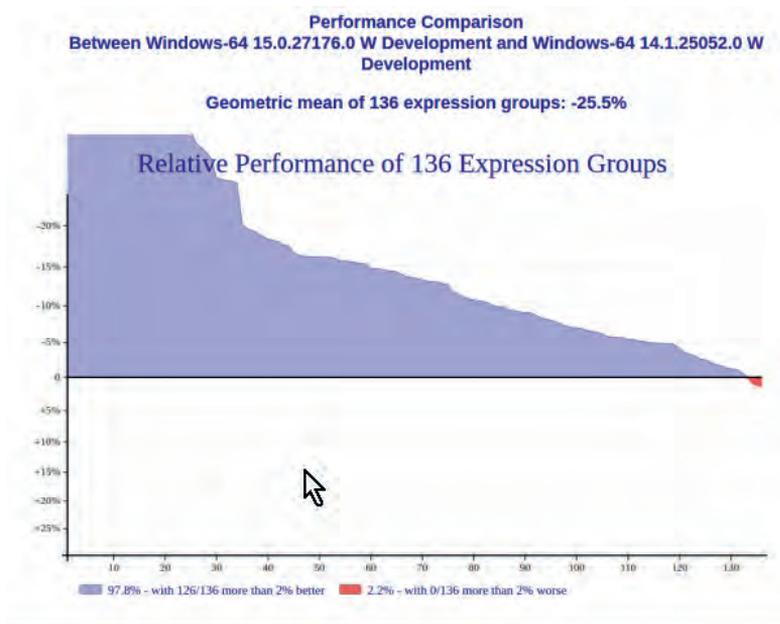
Performance

Ein Dauerbrenner, und die vielfältigen Projekte zeigen nun deutliche Früchte. Sehr beeindruckend zeigen das immer die Performance-Charts, welche die Performance-Verbesserungen zwischen den unterschiedlichen Versionen beim Abarbeiten einer Test-Suite von 136 Gruppen von 13.659 Ausdrücken analysieren. Mit V15 ergab sich im geom. Mittel eine Laufzeit-Verbesserung von 25%, umgekehrt gab es bei 2.2% der Funktionen eine Verschlechterung, die aber unter 2% lag. Insgesamt also ein deutlicher Performance-Zuwachs, der so auch von vielen Nutzern bestätigt wurde. Seit längerer Zeit hat Dyalog ein Projekt mit Aaron Hsu von der Universität Indiana, der daran arbeitet, dynamische Funktionen (d-fns) zu kompilieren und vom

Grafikprozessor des PC ausführen zu lassen. Nachdem das lange eher wie Themen von theoretischer Bedeutung erschien, bemühte er sich im letzten Jahr verstärkt darum, den praktischen Nutzen herauszuarbeiten. Das Potential ist beeindruckend, z.B. konnten beim Inner-Product Verbesserungen um Faktor 300-400 (!) beobachtet werden. Interessanter ist aber der Gesamteffekt, der in Bezug auf eine Applikation erreicht werden kann. Hier führte Aaron u.a. die ANN (Artificial Neural Networks) als Beispiel an, wo eine Verbesserung um Faktor 20 erreicht wurde.

Veli-Matti Jantunen: "The journey of an APL2 Bigot to Dyalog World"

Veli-Matti hielt eine Art Rückschau auf sein bisheriges APL-Leben und erzählte, wie er APL kennengelernt hatte und dann zu einem APL2-Anhänger wurde, bis er durch berufl. Veränderungen nach 12 Jahren den Umstieg auf Dyalog 6.3 beginnen musste. Sein erster Eindruck damals: das GUI und die Entwicklung von GUIs gefielen ihm, ebenso Tracer, Editor und die Möglichkeiten zur Excel-Anbindung, er störte sich an den exotischen Tastenkürzeln im Editor und vermissten Funktionen für den Umgang mit native Files. Als richtigen "Kulturschock" empfand er, dass seine Lieblingssymbole \uparrow , \Rightarrow , \Leftarrow , ϵ und \equiv sich auf einmal anders verhielten und der Interpreter einige Ausdrücke anders auswertete, z.B. musste er Ausdrücke wie `1 2 'ABC'[2]` in Dyalog umschreiben zu `1 2 ('ABC'[2])`. Nachdem er diese Hürden überwunden hatte und damit begann,



die Möglichkeiten zu erforschen und auszunutzen, nahm die Begeisterung zu (ich verzichte hier auf die Aufzählung aller herausgestellten Punkte – er war schlicht zu viel!). Mit Screenshots der verschiedenen Versionen der von ihm betreuten Software wurde dann deutlich, wo und wie die neuen Features der einzelnen Dyalog-Versionen sich auswirkten.

John Daintree: "The Airfix-Model: Programming from a Kit Of Parts"

Bei den Vorträgen von John Daintree darf man immer auf Überraschungen gefasst sein, dieser Titel klang aber zu langweilig und ernsthaft, um Wesentliches zu erwarten. Dachte ich.

John nahm uns mit in seine beschaulichen Kindheitserinnerungen – insbesondere an den Bau von Modellflugzeugen. Er erzählte, wie ihm dann irgendwann bei der Arbeit auffiel, daß das, was er nun tat, ja gar nicht so anders war: eigentlich, so meint er, nimmt man nun auch Einzelteile (Bibliotheken), die andere entwickelt haben und verbindet sie mit etwas Klebstoff (eigenem Code) zu etwas Neuem.



Und so begann er dann zu überlegen, was denn die verschiedenen Einzelteile wären,

die ihm zur Verfügung stünden. Klar, Dyalog, macOS, der Raspberry Pi und die RIDE-Umgebung. macOS und Raspberry Pi haben beide ja in gewisser Weise "Unix unter der Haube". Hm, war da nicht noch ein Betriebssystem, das irgendwie auch auf Unix aufsetzt? Ach ja, Android! für die Entwicklung unter Android gibt es mehrere Ansätze, u.a. auch ein Objektmodell, das auf Java aufsetzt. Java...? Ach ja, für die Java-Anbindung hatte John mal eine Bridge-Komponente entwickelt. Noch ein Baustein. Dann erinnerte er sich die Begegnung mit einem Anwender, der ihn darauf ansprach, ob er wohl dieses Jahr wieder einen Vortrag mit seiner CD-Datenbank ("CDDDB") machen würde. Der letzte Baustein! Und so nahm er all diese Bausteine mit in seine Werkstatt – und kam wieder mit der CDDDB, die nun auf dem Android-Fernseher, Telefon, Tablet und sogar auf der Smartwatch nutzbar war!

Bis all das für uns "Fußgänger" nutzbar ist, wird es wohl noch ein paar Monate dauern – aber diesen Ausblick auf die Möglichkeiten habe ich als sehr motivierend empfunden!

Thomas Gustafsson: "The Calm Before The Stormwind" / Vking-Challenge

Der für mich beeindruckendste Vortrag! Thomas führte seinen Boot-Simulator "Stormwind" vor. Eine Dyalog-Anwendung, bestehend aus 694 Funktionen, 661,447 bytes und 22.335 Programmzeilen. Hunderte Kunden, einerseits aus dem Segment der "professional users" (Seenotrettung, Marine, wissenschaftliche Einrichtungen etc.) und private Nutzer



(235 – 385€, je nach Umfang Kartenmaterial). Man kann damit eine Bootsfahrt simulieren. Es können unterschiedlichste Bootstypen in verschiedenen Gewässern gefahren werden, dabei wird echtes Kartenmaterial verwendet, Tag/Nacht- und div. Klimasituationen simuliert etc. Die Software ist sehr flexibel konfigurierbar, es können unterschiedlichste Gaming-Geräte angeschlossen und verwendet werden, um zu lenken, zu beschleunigen usw.

Soweit vorhanden kann dann eben auch ein Simulator mit eingebunden werden, über denn auch die Bewegungen des Bootes erfahrbar werden. In meiner Probefahrt gelang es mir nicht, das Boot zum Kentern zu bringen – aber die Simulation des Bootes erschien meinem Magen sehr glaubwürdig! ;-) Typischer Bestandteil der Dyalog-Konferenz ist ja immer auch eine "Viking-Challenge", irgendeine Herausforderung, die an die Glanzzeiten der alten Wikinger erinnert. Dieses Jahr wurde die Herausforderung mit dem Stormwind-Simulator umgesetzt: es galt, einen in Seenot geratenen Freund zu retten und das Boot auf dem schnellsten Weg zu gegebenen Koordinaten zu navigieren und dabei allerlei Überraschungen zu bewältigen.

Dr. Stephen Jaffe: "Composition Based Modelling and Dyalog APL"

Dr. Jaffe ist bereits im Ruhestand und berichtete anlässlich des 50-jährigen Jubiläums über diese Anwendung von Dyalog APL in der Öl-Industrie (ExxonMobil). Raffinerien verarbeiteten ursprünglich immer "lokales Öl" der nächstgelegenen Quellen. In den 70er-Jahren kam es durch das Ölembargo zu einer Änderung des Geschäftsmodells und die Raffinerien suchten nach neuen Wegen: es wurden neue Ölfelder erschlossen (u.a. auch in Afrika), deren Öl dann per Schiff zu den Raffinerien transportiert wurde. So hatten die Raffinerien auf einmal unterschiedlichste Sorten Rohöl zur Verfügung und mussten entscheiden, welches Öl sie (zu welchem Preis) ankaufen sollten und welches Endprodukt daraus hergestellt werden sollte. Beim "Composition Modelling" geht es nun um Prozesse, die das Herz eines jeden Chemikers höher schlagen lassen, die der Chronist aber in Ermangelung notwendiger Fachkenntnisse gar nicht erst versuchen möchte, zu beschreiben. Während dieses mehrstufigen Vorgangs gibt es jedenfalls viele Ansprechpartner, Berechnungen, Analysen und Zuständigkeiten – und Stephen resümiert ganz nüchtern: "jeder kann sich mal irren, aber unsere APL-Modelle lagen immer richtig!". Im Ergebnis gelangt man dann zu einem "Rezept", wie das Rohöl zu behandeln ist, ob ein maximales Ergebnis zu generieren.

Das System wurde ca. 1987-2006 in Dyalog APL entwickelt. Aus Performance-Gründen wurden Teilfunktionen dann sogar in Fortran ausgelagert, wobei der Fortran-Code jedoch durch die APL-Software erstellt und ausgeführt wurde:

In Zahlen:	
9.800	Zeilen APL Berechnung in einem typischen Modell
9.800 57.000	APL GUI
124.000	Fortran
12	Entwickler (Chemiker)
200	Benutzer
200.000	Modellberechnungen/Jahr
44	Raffinerien weltweit
5.5	Mio Barrel = 8.744.301 Hektoliter Öl täglich verarbeitet
1	Mrd USD = geschätzter zum Jahresgewinn nur durch beste Ausnutzung der Rohöle!

Charles Brenner:
"The Joy of (Especially Dyalog) APL and Some Gripes"

Rechenzeit schlicht nicht möglich, wie er mit dem Pseudo-Code aus dem Screenshot zeigte.

Der forensische Mathematiker Charles Brenner (der seine Berufsbezeichnung selbst

```
:For GenoCombo :In 1(10×10)*15
  calculate Score(GenoComb) A 1000 calcs/second?
:EndFor A 10*27 sec > anything
```

erfunden hat) ist ein weiteres Beispiel für einen "domain expert", der mit APL sein Expertenwissen in eine Anwendung gepackt hat, die weltweit genutzt wird (sogar beim BKA eingesetzt wird). Charles, der seit 1967 mit APL arbeitet, hatte ursprünglich seine Anwendung mit APL*PLUS entwickelt und sie dann in den letzten Jahren nach Dyalog APL portiert. Die durch die Software gelösten Probleme kann niemand so gut und lebendig beschreiben wie Charles selbst (nach eigenen Angaben ist er auch noch ein "Standup APL-Comedian"). Soweit ich es verstanden habe, geht es darum, DNA von Tatorten abzugleichen mit der DNA von Verdächtigen, wobei er sich speziell auf Fälle mit verunreinigter oder gemischter DNA mehrerer Verdächtiger konzentriert. Hierbei eindeutige Nachweise zu führen, ist wg. der dafür notwendigen

Charles fand jedoch eine Herangehensweise, dennoch in endlicher Zeit verlässliche (und vor Gericht belastbare Aussagen) zu machen – und das auch noch schneller und mit größerer Sicherheit, als es alle Wettbewerber können.

Und weil er ja im Titel des Vortrags versprochen hatte, auch zu nörgeln, kamen dann auch ein paar Verbesserungsvorschläge:

- um eine neue Funktion zu erstellen, kann man ja in der Session den Funktionsnamen eintippen und darauf Shift+Enter drücken. Shift+Enter in einer Leerzeile sollte direkt den Funktionseditor öffnen, schlägt er vor.
- Ferner bemängelte Charles, dass i-beam **I** das linke Argument numerisch ist. Um zum Beispiel den zu verwendenden

Zufallsgenerator auszuwählen, schlägt er vor, statt 16807 im linken Argument einen String, etwa 'RandomNumber-Generator' anzugeben.

- Bei STSC gefiel ihm, dass `QFRDCI` im Ergebnis auch direkt einen lesbaren Timestamp ausgab – so fragt er "why do I have to work so hard?" ("Warum muss ich so schwer arbeiten?") und hat ein Utility `QFRDCI` geschrieben, welches dieses Verhalten nachbildet.

Und außerdem...



(Foto: John Scholes)

Sehr interessant fand ich auch die zahlreichen Rückblicke auf 50 Jahre APL mit vielen persönlichen Anekdoten (zum Beispiel wie Geoff Streeter in Frankreich Schwierigkeiten mit dem Zoll bekam, weil man ihm nicht glaubte, dass 10 damals brandneue 3.5-Zoll Disketten 40 GBP Wert waren und deutlich teurer als die viel größeren Magnetbänder), spannender Hardware, faszinierenden Frisuren und einem ganz anderen Umfeld.

Ein anderer Teilnehmer erzählte in seinem Rückblick, wie er einen jungen Studenten kennenlernte, der einen Investor suchte für sein Projekt zur Entwicklung eines APL-Compilers. Nachdem er den Studenten besucht hatte, abgeschreckt war von dessen Gejammere über die Schwierigkeiten mit Hardware etc. und der Unordnung in dessen Büro entschied er schließlich, nicht zu investieren. Der Student war Bill Gates!

Weiterhin...

waren da noch die Gewinner der "APL Problem Solving Competition", die inzwischen schon zum 8. mal von Dyalog ausgerichtet wurde. Die Preisträger der 3 Kategorien (Finanzen, Bio-Informatik, Allgemein) hielten jeweils Vorträge, in denen sie Ihre Lösungen präsentierten. Ich finde es motivierend, zu sehen, dass durch solche Maßnahmen tatsächlich neue Generationen angesprochen werden und trotz der großen Konkurrenz auf dem Markt der Computersprachen begeistert sind von APL. Im Zusammenhang mit "Begeisterung" muss ich unbedingt noch Alex Weiner erwähnen: ein "Electrical & Computer Engineer", für den es ganz natürlich war, sich mit Matrizen auseinanderzusetzen, weil man in der Elektronik eben alles als Matrix darstellen kann. Und weil man ja auch Computer darüber dann modellieren kann (er hatte das Buch eine gewissen K. Iverson gelesen...), schien es ganz natürlich zu sein, sich mit APL zu beschäftigen. Weil er sich mit APL aber als Hobby beschäftigen sollte, suchte Alex ein "cooles Thema" und kam schließlich auf die Idee, Grafiken zu verarbeiten – das sind ja auch Matrizen. So entwickelte er eine Funktion, welche zwei Grafiken

kombiniert. (Einfach auf <http://yhnjuiik.com/> selbst ausprobieren.) Sicherlich keine Raketenwissenschaft, cool. Viel begeisternder als das war die Begeisterung, mit der von APL sprach und alle Leute aus dem Bekanntenkreis gleich "konvertieren" wollte – immerhin einen neuen APLer hat er gewonnen :-). Um diese Begeisterung dann ganz praktisch weiterzugeben, verschenkte er selbstgemachte Ansteckbuttons mit seinem Namen und einem netten Kommentar wie "APL is great" usw. Diese Eigenwerbung machte sich bezahlt: während des Meetings bekam er von einem der Teilnehmer ein Jobangebot!

Die Italiener waren auch wieder da!

Roberto Minervini ist ein ehemaliger Entwickler von APL Italiana, der trotz aller Freude an APL viel lieber als Lehrer arbeiten wollte. Dabei hat er dann auch APL genutzt und darüber schon im letzten Jahr einen Vortrag gehalten. (Die Gewinnerin des letzten APL-Wettbewerbs hatte übrigens bei ihm Unterricht!) Die theoretischen

Gedanken, wie man Schülern etwas so Abstraktes wie Mathematik und APL nahebringen kann, führten schließlich dazu, daß er versuchte, mit Hilfe von Cartoons den Schülern den Zugang zu erleichtern. Um zu beweisen, dass das funktioniert, gab es zum Schluss des Vortrags einen Wettbewerb, bei dem die Teilnehmer im Publikum direkt mitmachen konnten und über eine auf ihren Smartphones oder Laptops aufzurufende Seite versuchen mussten, zu erkennen, welche APL-Idiome in verschiedene Cartoons dargestellt waren. Wer es mal ausprobieren mag, kann das am folgenden Beispiel tun (Lösung am Ende des Artikels).

Der Kenneth E. Iverson Award for Outstanding Contributions to APL

oder kurz: der "Iverson-Award", eine Auszeichnung, die vergeben wurde von SIGAPL, der "special interest group for APL" innerhalb der ACM – der amerikanischen Vereinigung for "computing machinery". Seit 1983 19mal verliehen, in 2007 aber

Assume ω is a vector. This picture represents an APL idiom, which one is it?



55

3 Answers

$\{\omega \wedge \neq v / \omega\}$
 $\{\wedge / \omega \times v / \omega\}$

$\{\wedge / \omega = 1 \oplus \omega\}$
 $\{(\uparrow / \omega) + \lfloor / \omega\}$

letztmalig. (u.a. gibt es die SIGAPL inzwischen nicht mehr). Anlässlich des 50-jährigen Jubiläums gab es Bestrebungen, diesen Preis wieder zu vergeben und so übernahm die SIGPLAN-Gruppe von ACM die Preisvergabe und schließlich wurde der Preis am Abend der 50-Jahrfeier verliehen als Zeichen der Anerkennung des unermüdlchen Einsatzes für APL an Gitte Christensen und Morten Kromberg.

Was noch fehlt

Vieles! Ich hatte ursprünglich vor, nur auf drei Vorträge näher einzugehen, doch bei der Auswahl merkte ich dann, dass dieser aber nicht fehlen darf und ich jenen auch nicht unerwähnt lassen kann, so wurde es dann doch mehr – doch ich bin noch immer nicht zufrieden mit der Auswahl. Noch immer habe ich viele wertvolle Vorträge und interessante Referenten nicht erwähnt, nicht gesprochen von großen Anwendungen, von denen ich erfahren habe usw. Ich möchte daher ausdrücklich hinweisen auf Dyalogs Website zur Konferenz <http://www.dyalog.com/user-meetings/dyalog16.htm>, wo die Tagesordnung einsehbar ist und alle Vorträge mit Links zu Video und Downloads versehen sind – eine tolle Möglichkeit, auch im Nachhinein mal reinzuschnuppern!

Fazit

Die Dyalog-Benutzertreffen sind seit vielen Jahren regelmäßiger Höhepunkt meines APL-Jahres und ich freue mich immer auf die Zeit dort. Selten gibt es Gelegenheit, so viel über andere Anwendungen zu erfahren, sich mit anderen Entwicklern

auszutauschen, die Neuigkeiten von Dyalog kennenzulernen und in Workshops zu vertiefen. Ich bin immer gerne dabei und freue mich schon jetzt auf #Dyalog17!

Videos

Ich habe einige (kurze) Videos aufgenommen, die evtl. noch einen besseren Eindruck vermitteln:

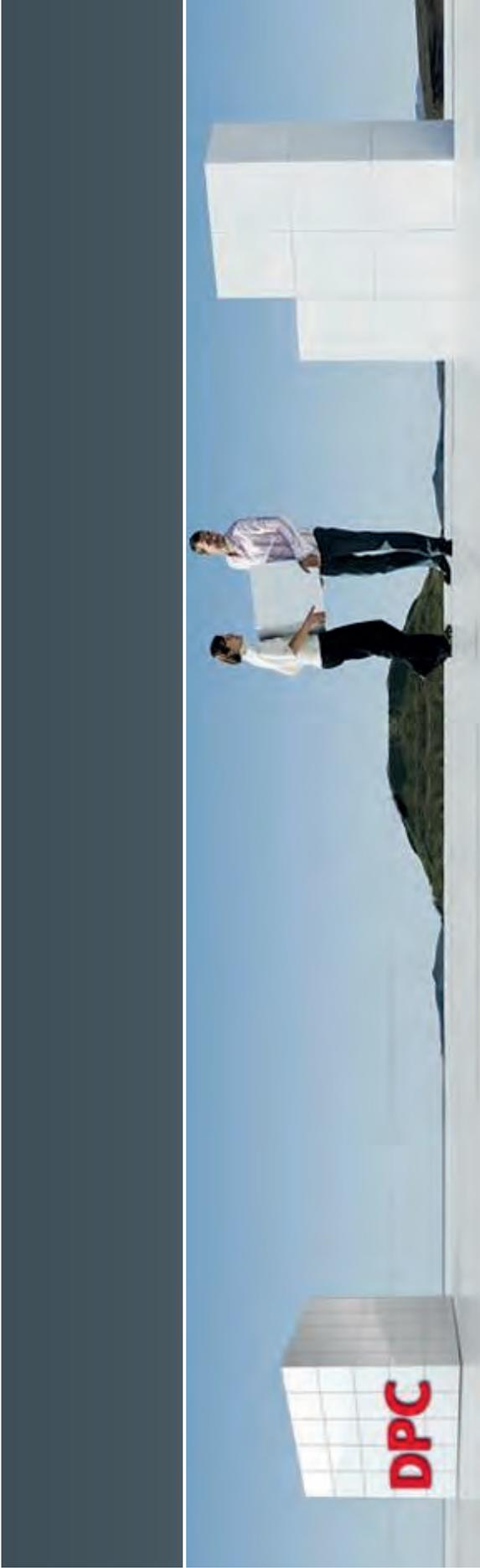
<https://www.youtube.com/watch?v=LKbnjVbed9Y> — Stormwind-Simulator in Action (etwas entfernte Sicht, damit man das "Gesamtwerk" mit dem Simulatorenbau sieht)

<https://www.youtube.com/watch?v=kE8xSiKsD8U&t=2s> — Verleihung des Iverson-Awards

<https://www.youtube.com/watch?v=8NK3UqVZX3Q> — "Ode to a haggis"

P.S: die Lösung zum Idiom-Cartoon: links unten :-)

Contact:
Michael Baas
APL Tools Group. Dyalog Ltd
DLS Software & Beratung GmbH
<http://www.dls-software.de/>
Email: michael@mbaas.de



50 years IBM APL

41 years IBM 5100

A personal computer that dominated the market
from 1975 to 1983

Gerald Dittrich APL-GSE IBM Böblingen 28.11.2016



5100 IBM Portable Computer

Introduced in September 1975, the 5100 Portable Computer was IBM's first production personal computer (six years before the best-seller [IBM PC](#)). The 5100 was intended to put computer capabilities at the fingertips of engineers, analysts, statisticians and other problem-solvers, but not for business purposes.



If the size and weight of the 5100 seems (25 kg) huge by today's standards, then the IBM 5100 was very slim compared to a late-1960's IBM computer with the equivalent capability. Such a machine would have been nearly as large as two desks and would have weighed about half a ton.

The 5100 was much more powerful than its predecessors, like [Altair 8800](#) (and much more expensive however—it was sold for between \$8975 and \$19975).



28.03.2017

SEITE 3



5100 IBM Portable Computer

It featured built in CRT display, keyboard, BASIC and/or APL interpreter and mass storage (tape drive).

It has also a much more advanced design: A microcoded 16-bit CPU (1.9MHz) executing an interpreter which in turn interprets a subset of the IBM 360 (or IBM/3) mainframe instruction set!

Available options:

- carrying case
- IBM 5103 printer, dot matrix, tractor feed, 132 column, 80 char/s bidirectional
- IBM 5106 external tape drive
- communication adapter
- serial I/O adapter



IBM5100 with additional monitor



5100 IBM Portable Computer – Programming languages

Depending on options installed, the 5100 can run the APL (see the [IBM 5100 APL reference manual](#)) and/or BASIC programming languages and can have 16K, 32K, 48K or 65K RAM (see the table below of twelve different models).

Memory	Programming language		
	APL	BASIC	Both
16K	A1 - \$8,975	B1 - \$9,975	C1 - \$10,975
32K	A2 - \$11,975	B2 - \$12,975	C2 - \$13,975
48K	A3 - \$14,975	B3 - \$15,975	C3 - \$16,975
64K	A4 - \$17,975	B4 - \$18,975	C4 - \$19,975

On a 5100 with both languages (APL and BASIC), the user's choice of language is selected by a toggle switch on the front panel.



5100 IBM Portable Computer - specifications

The 5100 has an internal five inch CRT, displaying 16 lines of 64 characters. Because the characters are so small, IBM provided a three-position switch to allow the user to select the display of all 64 characters of each line, or only the left or right 32 characters (interspersed with spaces).

Mass storage was provided by a 1/4-inch cartridge tape drive using DC300 cartridges to store 204 KB on 300 feet tape.

Instead of being written in the native microcode instruction set of the processor, the 5100's language interpreters (APL and BASIC) are written for more sophisticated *virtual machines*, and the microcode emulates those machines. This was done in order to economize on the amount of ROM (read-only memory) needed to implement the language interpreters, and possibly to speed the software development. The APL microcode emulates a subset of the System/360 instruction set, while the BASIC microcode emulates the System/3.



28.03.2017

SEITE 5



5100 IBM Portable Computer - specifications

The 5100 has an internal five inch CRT, displaying 16 lines of 64 characters. Because the characters are so small, IBM provided a three-position switch to allow the user to select the display of all 64 characters of each line, or only the left or right 32 characters (interspersed with spaces).

Mass storage was provided by a 1/4-inch cartridge tape drive using DC300 cartridges to store 204 KB on 300 feet tape.

Instead of being written in the native microcode instruction set of the processor, the 5100's language interpreters (APL and BASIC) are written for more sophisticated *virtual machines*, and the microcode emulates those machines. This was done in order to economize on the amount of ROM (read-only memory) needed to implement the language interpreters, and possibly to speed the software development. The APL microcode emulates a subset of the System/360 instruction set, while the BASIC microcode emulates the System/3.

28.03.2017

SEITE 7



5100 IBM Portable Computer – a look inside





28.03.2017

SEITE 8



IBM5110

- IBM5110 announced in 1978
- Still including tape drive
- Additional 2 Diskette drives (IBM5114, 1.2 Mbyte each, monster: large, heavy)
- Some functions got much faster e.g. Dyadic Iota



IBM5110 mounted on diskette unit

28.03.2017
SEITE 9

IBM5120

- Two years after launching the 5110, IBM introduced its 5120 Computer System in February 1980 as the lowest-priced IBM computer to date. A representative configuration — which included a main storage capacity of 32,768 characters of information, a 120 character-per-second printer and the BASIC programming language — could be purchased for less than \$13,500. Overall system prices ranged from \$9,340 to \$23,990.
- The 5120 Computer System featured the new desktop IBM 5110 Model 3 computer and two previously announced products: the IBM 5103 models 11 and 12 bidirectional, matrix printers; and the IBM 5114 diskette unit with up to 2.4 megabytes of direct access storage.



28.03.2017
SEITE 10



IBM5120



- IBM 5120 - announced in 1980
- 2 integrated disc drives (1.2 Mbyte) no longer tape drive
- Tape drive could be added as external unit
- Extremely heavy (some 45 kgs !)
- Larger screen but still not large enough



28.03.2017
SEITE 11



IBM51xx – „Amazing“ options and features

1 - Speed up the printer from 80 char/s to 120 char/s

- Price: 1000 Deutsche Mark
- IBM made only a small modification of the circuit board

2 - Add a serial interface

- Price: 1600 Deutsche Mark
- IBM changed the asynchronous cable by a Y-cable that supported both the asynchronous port and the serial port



28.03.2017

SEITE 12



IBM51xx at McKinsey

- McKinsey started 1975 with the first IBM5100 (at a cost of 80.000 Deutsche Mark)
- From 1975 to 1980 McKinsey had bought five machines (IBM5100, IBM5110, IBM5120)
- 60-100 days Pay-back period of investment of 80.000 DM:
 - One hour connection for timesharing service: 40 DM
 - One hour phone costs with acoustic couplers to Frankfurt: 69 DM
 - Timesharing CPU-costs depending on the amount of data (could be some 200-500 DM per day)



28.03.2017
SEITE 13



IBM51xx at McKinsey – integrated development process with host system

- APL-Programs were developed and tested with small data set on IBM5100
- Functions and data were sent to the timesharing computer (send_all functions were sending the whole workspace to the timesharing system)
- The APL-timesharing systems at that time were:
 - IBM Hamburg VSAPL , 200K workspace
 - Datema (Sweden) with VM-CMS up to enormous 16 MB (access point in Frankfurt, later in Duesseldorf)
 - IBM with VM-CMS in Duesseldorf , up to 16MB
 - IP-Sharp (access point Brussels ?)
 - APL-Plus (access point Duesseldorf)



28.03.2017

SEITE 14



IBM51xx at Gerald Ditttrich Consulting

Started my own career when I left McKinsey in Jan 1981

- Start with one IBM5120 (stock order machine 48K), increasing to five after 3 years
- Still working with timesharing

Small workspaces require special (sometimes amazing) programming (the maximum net workspace could not exceed 57K!)

- Rectangular looking functions with short variable or function names
- Run workspace : Functions were de-commented to save workspace, sometimes some guys even did a precompile work on functions (branching to a line number, no longer labels - nearly unreadable- but resulted in faster execution)
- File system to load and unload functions to save workspace
- Blocklooping methods for groupsumming



28.03.2017

SEITE 15



IBM51xx at Gerald Dittrich Consulting - some example projects

BMW motor production: simulation of 160 real-time hours

- **Production line : up to 200 stations in 15 sections**
- Stations: drilling, turning, deburring, milling, honing, lapping, sawing, grinding, washing and more
- Optimizing buffers between sections, optimizing maintenance strategy
- 160 real-time hours meant 70 hours on a IBM5120
- 20 simulations to do
- A try of 16 hours real-time on IPSharp resulted in costs of 1,500 DM
- 20 simulations of 160 real-time hours would mean 300,000 DM
- Decision: run simulations on 5 IBM51xx
- After some 12 days: very good results
- No system errors, no shutdowns , each simulation finished



28.03.2017

SEITE 16



IBM51xx in museums

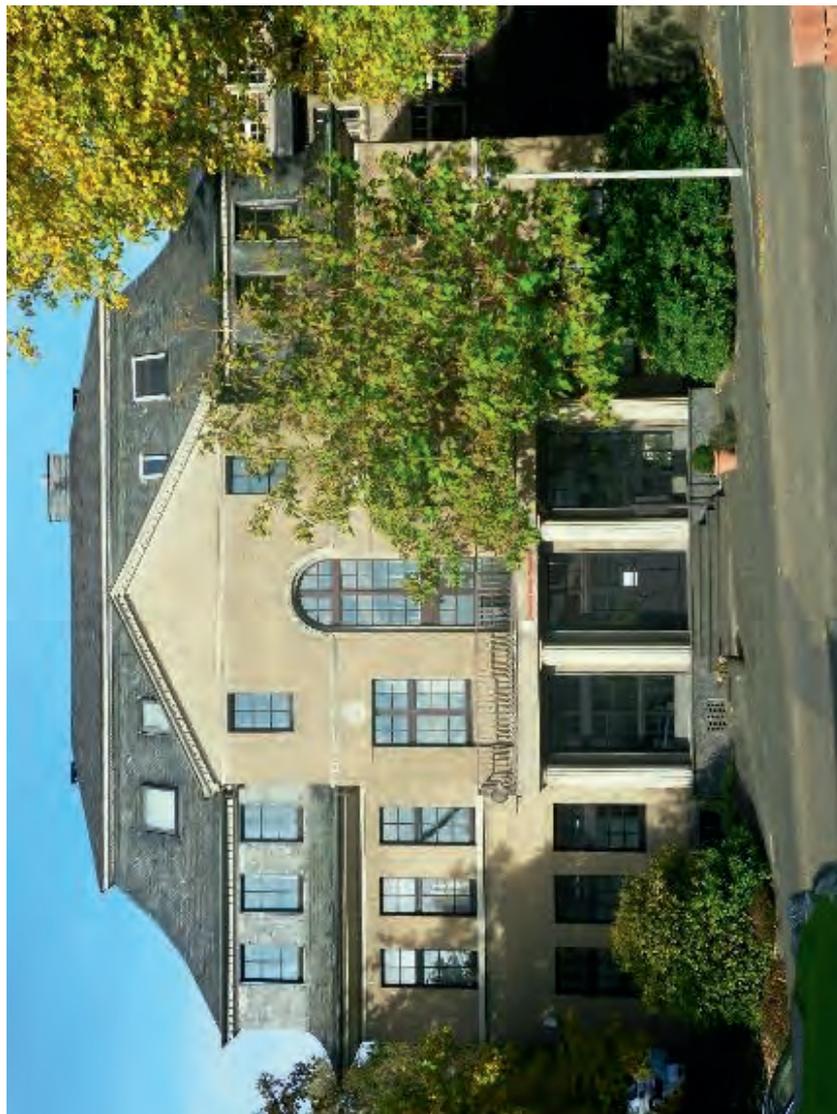
- IBM5110 seen 1993 in Technical Museum Kopenhagen
- IBM51xx seen 2010 at Technical Museum Berlin where you can also see the worldwide first computer Zuse (still working, kept alive by the son of Konrad Zuse)
- IBM museum at Stuttgart
- Private Museum Gerald Dittrich in Solingen : 3 machines IBM5100, IBM5110, IBM5120, printer IBM5103, diskette unit IBM5114, each machine is still working
- Other museums ?



28.03.2017
SEITE 17



Private Museum of Gerald Dittrich in Solingen Prinzenstr. 2





28.03.2017

SEITE 18



DPC in Solingen viewed from a quadrocopter



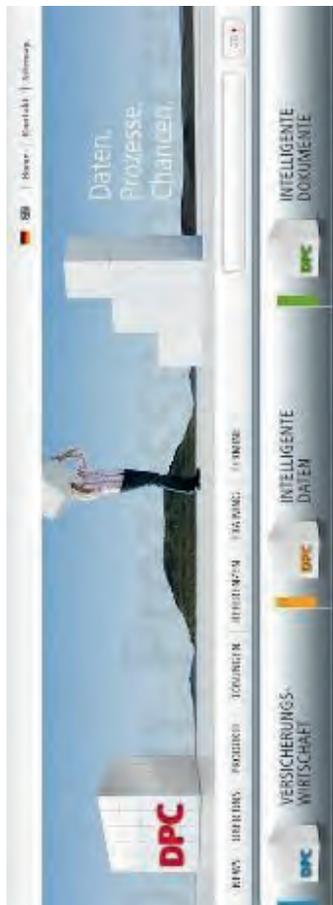


28.03.2017
SEITE 19



Intelligent data | Intelligent documents

Visit us !



■ www.dpc.de | info@dpc.de



■ www.dpc-software.de | www.dpc-software.nl | www.dpc-software.eu

APL-Journal

35. Jg. 2016, ISSN 1438-4531

Herausgeber: Dr. Reiner Nussbaum, APL-Germany e.V., Mannheim, <http://www.apl-germany.de>

Redaktion: Dipl.-Volksw. Martin Barghoorn (verantw.), FU Berlin, Zentraleinrichtung für Datenverarbeitung (ZEDAT), Fabeckstraße 32, 14195 Berlin, Tel. (030) 804 03 192

Verlag: RHOMBOS-VERLAG, Berlin, Kurfürstenstr. 15/16, D-10785 Berlin, Tel. (030) 261 9461, eMail: verlag@rhombos.de, Internet: www.rhombos.de

Erscheinungsweise: halbjährlich

Erscheinungsort: Berlin

Satz: Rhombos-Verlag

Druck: dbusiness.de GmbH, Berlin

Copyright: APL Germany e.V. (für alle Beiträge, die als Erstveröffentlichung erscheinen)

Fotonachweis Titelseite und Umschlagseite 4: Martin Barghoorn

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutzgesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen. Eine Haftung für die Richtigkeit der veröffentlichten Informationen kann trotz sorgfältiger Prüfung von Herausgeber und Verlag nicht übernommen werden. Mit Namen gekennzeichnete Artikel geben nicht unbedingt die Meinung des Herausgebers oder der Redaktion wieder. Für unverlangte Einsendungen wird keine Haftung übernommen. Nachdruck ist nur mit Zustimmung des Herausgebers sowie mit Quellenangabe und Einsendung eines Beleges gestattet. Überarbeitungen eingesandter Manuskripte liegen im Ermessen der Redaktion.



Allgemeine Informationen

(Stand 2016)

APL-Germany e.V. ist ein gemeinnütziger Verein mit Sitz in Düsseldorf. Sein Zweck ist es, die Programmiersprache APL, sowie die Verbreitung des Verständnisses der Mensch-Maschine Kommunikation zu fördern. Für Interessenten, die zum Gedankenaustausch den Kontakt zu anderen APL-Benutzern suchen, sowie für solche, die sich aktiv an der Weiterverbreitung der Sprache

APL beteiligen wollen, bietet APL-Germany den adäquaten organisatorischen Rahmen.

Auf Antrag, über den der Vorstand entscheidet, kann jede natürliche oder juristische Person Mitglied werden. Organe des Vereins sind die mindestens einmal jährlich stattfindende Mitgliederversammlung sowie der jeweils auf zwei Jahre gewählte Vorstand.

1. Vorstandsvorsitzender

Dr. Reiner Nussbaum
Dr. Nussbaum gift mbH, Buchenerstr. 78, 69259 Mannheim, Tel. (0621) 7152190.

2. Vorstandsvorsitzender:

Martin Barghoorn
Zentraleinrichtung für Datenverarbeitung (ZEDAT), Fabeckstraße 32, 14195 Berlin, Tel. (030) 804 03 192
eMail: Barghoorn@zedat.fu-berlin.de

Schatzmeister

Jürgen Beckmann
Im Freudenheimer Grün 10
68259 Mannheim
Tel. 0621 7 98 08 40,
eMail: JBecki@onlinehome.de

Beitragsätze

Ordentliche Mitglieder:

Natürliche Personen 32,- EUR*
Studenten / Schüler 11,- EUR*

Außerordentliche Mitglieder:

Jurist./natürl. Pers. 500,- EUR*

* Jahresbeitrag

Bankverbindung

BVB Volksbank eG Bad Vilbel
BLZ 518 613 25
Konto-Nr. 523 2694

Hinweis:

Wir bitten alle Mitglieder, uns Adressänderungen und neue Bankverbindungen immer sofort mitzuteilen. Geben Sie bei Überweisungen den Namen und/oder die Mitgliedsnummer an.



Einzugsermächtigung

Mitglieds-Nr.: _____

Ich erkläre mich hiermit widerruflich damit einverstanden, daß APL Germany e.V. den jeweils gültigen Jahres-Mitgliedsbeitrag von meinem unten angegebenen Konto abbucht. Mit der Datenübermittlung an das oben genannte Kreditinstitut bin ich einverstanden. Einen eventuell bestehenden Dauerauftrag habe ich bei meiner Bank gelöscht.

Bankbezeichnung: _____

BLZ: _____ Konto-Nr.: _____

