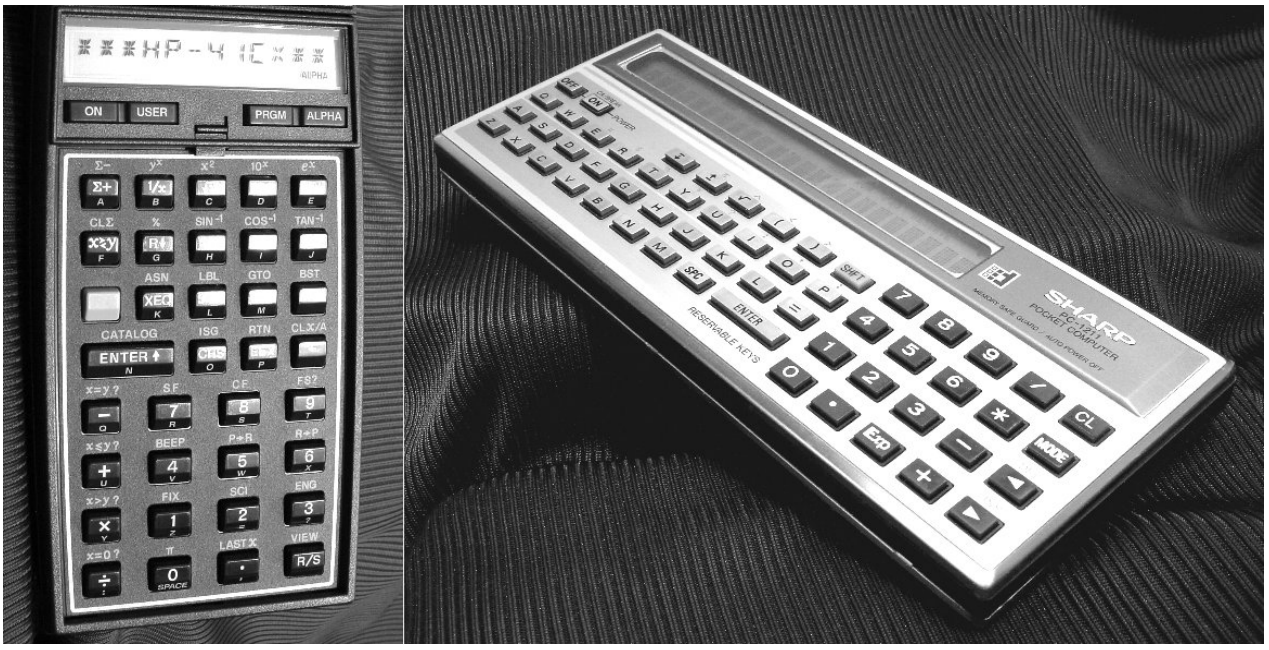
Know Thy Foe: A New Contender

Valentín Albillo (HPCC #1075)

Prelude

25+ years ago, the 80's just beginning, I already was a seasoned HP fan and user, with extensive experience with the **HP-25**, the **HP-67** and the brand-new, awesome **HP-41C**, synthetics and all. Yet I was always longing for more power and better functionality. The **TI** machines absolutely never caught my fancy, they were obviously sorely lacking in built quality, and their programming paradigm was dreadful and awkward to say the least. **RPN** in contrast was a clear sky and refreshing breeze when compared to the horrors of blind-programming **TI**'s **AOS**.

Back then a friend of mine, who had just recently bought an **HP-85** computer, also owned an **HP-41C**. But after he became used to the **HP-85** and its **BASIC** language, he began to find the **HP-41C** unfriendly: a high-level language such as **BASIC** had little in common with the "machine language" style of **HP-41C** programming. He longed for a handheld using a programming language more akin to that of his **HP-85**. The solution: he sold the **HP-41C** and bought instead the just introduced BASIC-programmable **SHARP PC-1211 Pocket Computer**. That way he needed not change his programming style when switching from his computer to his calculator and vice versa thanks to their high degree of compatibility for most casual, simple programming tasks. Lots of publicly available programs written in some **BASIC** dialect would usually convert easily as well, and he could much more easily switch from one **BASIC** dialect to another than to **RPN**. By getting rid of the **HP-41C** and getting instead the **PC-1211**, he fulfilled his desire to have a similar programming environment both in the office and in the field.



Enter The Foe

To my utter delight, he lent me his **PC-1211** for a week, and I came to like the machine *a lot*, eventually buying one myself. It was also available in the US rebranded as the **TRS-80 Pocket Computer**, which was the exact same machine except for the **TRS** label. Later on, when other **TRS** models were introduced, it was retrospectively known as "Model 1" or **PC**-1, to distinguish it from them.

The **SHARP PC-1211** is a very slim, compact handheld, very pretty looking, certainly much prettier than the **HP-41C** which looked clumsy and 'ancient' in comparison. It has an all-metallic body (unlike the **HP-41C**'s plastic body), and a 24-character dot-matrix yellow LCD display (unlike the 12-character, segmented grey LCD of the **HP-41C)**, which results in double the info displayed at once and much better looking alphanumeric characters than the ones the 41C could display using just 14 rectilinear segments. See for yourselves:

The **PC-1211** was also quite inexpensive, as well. The price, including both the handheld proper as well as the cassette interface was less than half that of a bare-bones **HP-41C**. Adding the much needed card reader to the **HP-41C** would mean duplicating your investment and still have bare-bones memory, namely 445 bytes maximum for programs and data, which were insufficient to the point that most moderately complex programs for the **HP-67** would not fit and/or run. On the other hand, the **SHARP PC-1211** handheld featured 1,920 bytes for programs/data right out from the box, more or less equivalent to an **HP-41C** plus 3 RAM modules (the maximum you could fit at the time if also using the card reader), which gave ample room to develop complex programs.

The **PC-1211** has 'continuous memory', like the **HP-41C**, so that programs, data, key assignments, and operating modes are all preserved when the machine is turned off, and it uses standard silver oxide button batteries that last very long, many months of heavy use, a definite advantage when compared to the difficult-to-

find N-size batteries the **HP-41C** needs, which would last only so many hours, much less if also using the card reader, even if sparingly. That gave the **PC-1211** utmost portability, without ever having to worry about a dead machine in the middle of some critical assignment.

As for the keyboard, though not nearly as good as **HP** keyboards of the time (none were), it was pretty adequate as well, with the keys providing some tactile feedback and never failing to register. It was also pretty durable, mines still working perfectly after 25+ years.

Heresy !

Used in manual calculations, the **PC-1211** is a delight. I never liked **TI**'s **AOS**, because you got lost using brackets and functions, but this was mainly due to the fact that you couldn't see the computations in the display, just the results. Once a sequence of operations was entered you had no access at all to the operands and operators, you simply pressed the **[=]** key and got a result, no way to know if it was correct or not, except by repeating the calculation and seeing whether the new result agreed with the former. Which is more, even if you were sure the result was wrong, there was no way to know where the mistake had taken place nor any way to correct it and try again without reentering it all from scratch.

That being so, it's easy to see why everyone knowledgeable with both systems would prefer **RPN** hands down: no messy operation, you have complete access to intermediate results, less keystrokes are needed, and parentheses definitely aren't. There's also consistency, which **AOS** actually lacked as some operations weren't entered as per true algebraic logic but in postfix notation, such as computing the sine of 45: you'd enter 45, then press **SIN**. But even back then in the early 80's I was fully aware that **RPN** wasn't without blame either. First, the 4-level stack is insufficient to casually attack most complex problems left-to-right so you need to stop and think about the proper order of operation in order to successfully compute the expression without ever overflowing the stack with intermediate results, and

that would usually entail some necessary reordering prior to pressing any keys. Which is more, *you* would do this preliminary work, not the machine. Of course, with a little practice this is no great difficulty and becomes second-nature, but the fact remains that I felt at the time that the ideal calculator should *not* mandatorily require *any* such effort *on the user*, and with a 4-level stack, thinking ahead was required no doubt. There was also the fact that, like it or not, most computations are written down in paper in standard algebraic form, and to compute them using **RPN** requires, again, some translation work on the user's part.

But lo and behold, here we had a small, slim programmable handheld that would accept BASIC expressions from the command line to manually perform computations, and of course BASIC uses full algebraic hierarchy to do the work, so that you can enter expressions as written in true algebraic form, where true means that if you want to compute Sin(45) you enter **SIN 45**, not **45 SIN**. This way, **SHARP** consigned to oblivion most drawbacks of **AOS**-like systems: now you wouldn't get lost while entering data and operators because you're seeing *everything* in the display as you're entering them, and *nothing* is evaluated till you press **ENTER**; you've also got 24 display positions for your expression and if longer, the display will automatically scroll as needed. The command line could be up to 80 bytes long, which translated to more than 80 characters as all operations were efficiently *tokenized* so that **SIN** is 3 characters but only 1 byte in the command line and counts as a single display position against the 80 available.

You could always use the cursors (which repeat if held pressed) to edit the expression, either while entering it or even after having computed it, as you could recall *the entire expression* just computed to the display, for revision or editing. Compare that to the **HP-41C**, where you just had **LASTX** to recover *the last number* in **X**, but none other operands or operators. Upon evaluation, all errors would be reported as an error code and you could recall the whole expression back to the display, with the cursor blinking at the exact point where an error was first detected. That way, you could most easily correct any mistake, being able to delete, insert, or replace any characters in any part of the expression, pressing **ENTER** to re-evaluate it when finished with the corrections. Nothing like this in the **HP-41C**.

Besides, this capability of recalling whole *already computed* expressions to the display was immensely useful to perform repeated computations with different data and play what-if scenarios without ever requiring programming. Also, you could compute several expressions at the same time by simply separating them with commas, and for further convenience, the result of a computed expression can be used in another by simply typing the new expression immediately; say, to compute 5 plus 6, see the result, and then add 7 to it, you would simply press: **5+6 ENTER** (see **11**) **+7 ENTER** (see **18**). As an example, suppose you want to manually compute the roots of some quadratic equations. You would type in:

```
A=1, B=5, C=-6, (-B+√(BB-4AC))/2A
```

then press **ENTER** to evaluate it to 10-digit accuracy (12-digit internally). Though it's a pretty simple expression, you can see that you're in fact doing three variable assignments plus computing the actual value for the root using those variables. You can also notice that implied multiplication *is* allowed (where **BB** means **B*B**, **4AC** means **4*A*C**, and **2A** means **2*A)**, saving lots of keystrokes and time, and further notice that the algebraic hierarchy is advanced enough that implied multiplication is performed *before* anything else, so that 2*A is computed before attempting division by it !.

This example is also perfect to demonstrate the powerful editing facilities. After the root is computed and displayed, we can compute the *other* root by simply recalling the expression back to the display, placing the cursor over the **+**, press **–** over it, then **ENTER**. The edited expression is evaluated and the second root appears. Which is more, to solve other *different* equation, recall the expression back again, then edit the values assigned to **A**, **B**, and/or **C**, then **ENTER**. You'll agree it's clearly astounding what can be done with the **SHARP PC-1211** under pure manual calculations, without programming at all. This can't be done with an **HP-41C** unless programmed.

Hierarchy rights

Another strong point of the **SHARP** is its full algebraic hierarchy. Not only does it feature true algebraic notation (e.g.: **SIN 45**) and implied multiplication (e.g.: **BB-4AC**), but it also does allow intermixing logical computations (e.g.: **5*(A=0)+EXP(J)*(F$=G$)-7*(B>=C+1)**) and it has an internal 8-level stack for intermediate results and a 15-level stack for functions and operators, so that in practice nearly all expressions can be entered left-to-right, as written. The well-known Mach number example, which **HP** usually gave as an example of the power of **RPN**, can be entered left-to-right with plenty of internal levels to spare, while in the **HP-41C** and other 4-level **RPN** models you *must* decide in advance *where to begin* lest you'd find yourself losing an intermediate result from the top of the stack. The ability to enter expressions left-to-right not only saves effort on the part of the user, but it saves time as well, as none is wasted considering what the correct evaluation order should be to avoid losing data.

This is not the only time-saving feature. Another obvious way to save time and effort is reducing the number of keystrokes needed to enter expressions to a minimum. In this regard, **RPN** was always considered to lead the pack, usually needing less keystrokes than any **AOS** version. But the **SHARP PC-1211** algebraic logic gives it a run for its money, with such niceties as implied multiplication (which saves one **[*]** keystroke per factor: **5*A*B*C*D** can be evaluated in just 6 keystrokes, **5ABCD** plus **ENTER**), parentheses not being needed for function arguments (so **COS(-2*X*Y)** can be entered as **COS -2XY**), and closing final parentheses being unnecessary (so that **3*(A+2*(B+6*(C+7))))** can be evaluated as **3*(A+2*(B+6*(C+7 ENTER** and all pending parentheses will be automatically closed). These clever measures result in less keystrokes, thus faster manual computations and program bytes saved as well.

Data & indirection

Like the **HP-41C**, the **SHARP PC-1211** also features dynamically allocated RAM, which is automatically partitioned between programs and data. A maximum of 204 registers (called 'variables') can be allocated (versus 63 for the bare-bones **HP-41C** and 191 for an **HP-41C** with 3 RAM modules plugged-in), 26 of which are permanently allocated to variables **A-Z** (numeric) or **A$-Z$** (string), the rest being accessible as array elements (say **A(138)** or **A$(26+10*I)**). When storing strings, each variable or array element can store up to 7 characters (versus 6 characters per register for the **HP-41C**). The equivalent of **STO** is a simple assignment such as **A=SIN 30**, the equivalent of **RCL** is just using the variable's name, so that you'd press **A [ENTER]** to display the contents of variable **A**. Indirection is achieved by array indexing, so that you'd use **A(N)** to access the N-th element of array **A**. Of course you can use more complex expressions such as **A(10*I+J)-A(I-J)**, which would require a bit of storage arithmetic and two registers in the **HP-41C** (or else doing the arithmetic on the stack, then using **RCL IND X** a couple of times). Also, you're not limited to a single level of indirection, so you can use expressions like **A(A(A(X)))** which in the **HP-41C** would be equivalent to **RCL IND IND IND X**, say, if it were at all possible.

Key assignments

An extremely useful feature of the **HP-41C** is the capability of assigning frequently used functions and programs to keys in **USER** mode, thus avoiding the need to continually having to spell out function names. The **PC-1211** allows you to assign functions and programs to keys, but the mechanism is quite different. First of all, assigning functions to keys does not consume program or data memory, but rather a separate 48-byte area is used instead, which is enough for a theoretical maximum of 24 key assignments. Secondly and most important, unlike the **HP-41C** which is limited to 1- or 2-byte assignments, in the **PC-1211** you can use a variable number of bytes for a single assignment, from a minimum of 2 up to a maximum of 47. This allows you to use assignments as typing aids, and for instance you can have the sequence **v(XX+YY)** assigned to the (shift)**A** key. Then, just pressing (shift)**A** would enter that sequence in the display, ready for evaluation at the press of **ENTER**, or forming part of a complex program line. This is very convenient and useful in practice and saves a lot of typing. A final feature is that the very same key can be assigned simultaneously to an expression or typing aid and a program. Programs are "assigned" as follows: the program line

```
10 "Z" REM *** POLYNOMIAL SOLVER ***
```

both gives a title to the program and specifies that pressing (shift)**Z** will automatically start execution at that line. You can have multiple entry points assigned this way, so you can start execution of different programs or different sections within a single program at the touch of the appropriate shifted key. This provides the equivalent of assigning global labels to keys in the **HP-41C**.

Programming the beast

While the **HP-41C** does use a fairly low-level 'keystroke programming' **RPN** style, the **SHARP PC-1211** was the very first small handheld in the world to include a high-level programming language, namely a fairly decent version of **BASIC**. Due to its pocket size and prize (thus ROM and RAM limitations) its **BASIC** is somewhat limited in some aspects, and it isn't particularly fast, but all in all it's a decent implementation, with many interesting extensions, 12-digit internal computations, and a healthy number of mathematical functions and features.

The first thing a fledgling programmer would notice is that programming in **BASIC** is quite comfortable. The **HP-41C**'s **RPN** style of programming means that the programmer must take care of all details. For instance, if a loop is to be programmed, you must take care to *construct the control register* (**bbbbb.eeeii**), include a branching point (**LBL 01**), decide whether it is to be incrementing (**ISG**) or decrementing (**DSE**), include the test condition (**X=Y?**) and finally a branching instruction (**GTO 01**). The resulting code isn't particularly readable or elegant. On the other hand, you can simply set up a **FOR...NEXT** construct in the **PC-1211**, which will automatically take care of all the details without explicitly requiring most of the elements mentioned. Which is more, **FOR...NEXT** loops can be nested, of course. The simple construct, which fits in a single line:

```
5 FOR I=1 TO 9:FOR J=1 TO 7:A(I)=3*I+J:NEXT J:NEXT I
```

is trivial to design and code without thinking at all, while the **HP-41C**'s equivalent construct would be a many-line affair, with two labels, two **GTO**'s, two comparisons, plus instructions to set up both incrementing indexes. Inputting data to a running program is also particularly powerful. A line such as:

```
10 INPUT "ENTER WIDTH",W,"ENTER LENGTH",A(30)
```

would alphanumerically prompt for the value of Width, which the user can then enter as a single number *or any arbitrary expression* which will be evaluated on the fly, then entered into variable **W**; you'll be prompted for Length, and your input will be evaluated and entered into the 30th element of array **A**. Again, not only is this much simpler and cleaner than the equivalent **HP-41C** sequence, but actually *you're not guaranteed that you can perform an arbitrary computation in the **HP-41C** as your input*, because perhaps the stack can't be disrupted at that particular point, as it frequently is the case. This also applies to stopping the program at an *arbitrary* point to do some manual computation, then resume. That you can't do in the **HP-41C** because you'll disturb the stack thus probably forfeiting any further resumption of the program. No such problem on the **SHARP PC-1211**, where you *can* stop a running program *at any point*, perform any manual calculation, then confidently resume program execution. You can inspect the values of variables and alter them if desired before continuing.

Program lines (which on the **HP-41C** can hold a single instruction or a number) can be up to 80 bytes in length, multiple statements being separated using "**:**". Statements can be entered with arbitrary spacing, upon pressing **ENTER** they will

be *tokenized* to save space and time (so that **SIN** is 3 characters when typed in but just 1 byte in program memory) and automatically spaced properly so that, for instance, should you type **10IN P UTV**, it will appear as **10:INPUT V** as soon as you press **ENTER**. Nothing like this in the **HP-41C**, of course.

Program editing & debugging

Programs can be listed and viewed line by line, scrolling the line if too long to fit the 24-character display. Lines can be inserted in any order whatsoever with arbitrary numbers (they'll be put in their proper order automatically), can be deleted by simply entering their number and pressing **ENTER**, and can be edited by inserting, deleting, or replacing characters using the cursor and edition keys. As you can edit the line number itself, it's very easy to quickly enter a number of very similar lines by merely *duplicating* them and then editing the duplicates.

Also, apart from speeding up program entry by using typing-aid assignments as discussed above, most keywords do have an *abbreviated form* (similar to the **HP-75C**) so that, for instance, you don't need to type **INPUT** in full but merely **I** and a dot. As soon as the statement is entered or executed, it will be completed to **INPUT**. Likewise you have **G.** for **GOTO**, **N.** for **NEXT**., etc. This is quite intuitive, very easily mastered, and makes program entry much quicker. It would be great if such a mechanism existed in the **HP-41C**, so that you could enter **PROMPT** by simply pressing **XEQ ALPHA PR.** , say, with the dot both converting **PR.** to **PROMPT** and taking us out of **ALPHA** mode. When you combine this mechanism with implied multiplication, automatic closing of final parentheses, automatic closing of final quotation mark (for strings), and multi-statement lines, it's clear how all these features work sinergically to help save lots of keystrokes and work, thus speeding program entry and manual computations greatly. It's ergonomic design at its best.

As for debugging, programs can be single-stepped. If an error occurs during execution, an error code is displayed and upon pressing the cursor key, the program line where the error was encountered will be displayed, with a blinking cursor over the precise offending statement, so that you can correct it at once.

Conditionals and branching

The **GOTO/GOSUB** branching instructions have been extended to allow *computed destinations*, which provides for indirect branching and then some:

```
GOTO 10, GOSUB 20*SIN(A+B)*EXP X, GOTO "DOIT", GOSUB R$
```

This is a very powerful extension, absent from both the **HP-85** and even the **HP-71B** (you had the much simpler **ON..GOTO/GOSUB** statements instead), which is similar but more powerful than **GTO IND X** in the **HP-41C**, say, where the **X** register contains the computed destination, either a numeric or an alpha label.

Conditional are also another strong point when compared to the **HP-41C** meager equivalents. In the **HP-41C** you can do very simple tests such as **X<Y?** or **X=0?,** or test a flag, then if the test is met you have *a single program step* to decide what

to do, usually branching to some label or subroutine. On the other hand, the **PC-1211** includes full conditional capabilities, like this test demonstrates:

```
IF A+C<=SIN(XY)/A(3+J) BEEP 1:PRINT "OK":N=N+1:GOTO 80
```

which would require a lot of code to mimic in the **HP-41C**. The **PC-1211** allows **FOR..NEXT** loops after the **IF** true clause (not possible in the **HP-85** or **HP-71B**) or even another nested **IF** (again, not possible in those machines).
Tests requiring some intrincate combination of the standard boolean operators **AND/OR/NOT/XOR/EXOR** are easily accomplished in **SHARP**'s **BASIC** dialect. For instance, suppose you want your program to branch to line 100 if **A** equals **B** or if, simultaneously, **C** is greater than **D** and **H** equals **B+D**. The line:

```
IF (A=B)+((C>D)*(H=B+D)) GOTO 100
```

accomplishes this effortlessly. Again, this would be much more cumbersome in the **HP-41C**, and would require some extra labels and branching.

Input/Output

You can save and load programs and data to cheap, standard audio cassette tapes, both manually and *under program control*, no need for a very expensive card reader and low-capacity magnetic cards. A single inexpensive C-60 audio tape will store dozens of programs. Additionally, programs and data can be merged with the current contents in RAM, which means you can run very large programs, each segment automatically loading and merging the next, unattended. As for printing, you can print items to the display or the impact printer, with provision for formatted output (similar to **IMAGE** in the **HP-75C** or **HP-71B**), like this:

```
PRINT USING "#.##";C+SIN(D);",ID=";A$(I);USING "#.#^";J
```

Documenting it all

The **PC-1211** comes with 3 manuals: the ***Owner's Handbook*** and the ***Beginner's Guide to BASIC*** are well written, discuss all aspects of the machine's operation and capabilities in a tutorial style, and feature many good examples, including useful exercises and their solutions. You can certainly start from scratch and painlessly become a proficient user and programmer, much like the traditional quality classic manuals from **HP** did, the outstanding **HP-41C** ones among them.
But the **PC-1211**'s ***Standard applications*** manual beats the ridiculously tiny and trivial **HP-41C**'s equivalent hands down. It is a 300+ pages book containing *over 130 programs*, all of them fully documented including algorithms and a wealth of examples. Unlike the ones featured in the **HP-41C**'s ***Standard applications*** book, these are non-trivial programs in all kinds of disciplines such as engineering, math, statistics, finances, even games. You'll find here Linear Systems Solving (up to 11x11), Matrix Inversion, Root Solving, Structures, Distributions, Regressions, TVM, Mortgages, etc. Not only will they prove invaluable to learn advanced programming techniques, but *they're useful by themselves right out of the book.* This manual is a real model for other manufacturers to follow, **HP** included.

Hand to hand: a casual duel

Let's now have a look at how both machines deal with a couple of rather simple programming tasks. This will provide a good taste of their respective programming paradigms and architectures.

Task 1: To find 3-digit integers equal to the sum of the cubes of their digits

The **PC-1211** program is an absolutely straightforward, 2-line affair:

```
10 N=100: FOR A=1 TO 9: FOR B=0 TO 9: FOR C=0 TO 9:
   IF AAA+BBB+CCC=N THEN PRINT N
20 N=N+1: NEXT C: NEXT B: NEXT A
```

which is 63 bytes long. The equivalent **HP-41C** program would be:

```
01 *LBL "NNN"   14  INT      27  RCL 00
02  100         15   3       28  X=Y?
03  STO 00      16  Y^X      29  STOP
04  1.009       17  RCL 02   30   1
05  STO 01      18  INT      31  ST+ 00
06 *LBL 01      19   3       32  ISG 03
07  .009        20  Y^X      33  GTO 03
08  STO 02      21  RCL 03   34  ISG 02
09 *LBL 02      22  INT      35  GTO 02
10  .009        23   3       36  ISG 01
11  STO 03      24  Y^X      37  GTO 01
12 *LBL 03      25   +       38  CLX
13  RCL 01      26   +       39  END
```

which is obviously larger, more complex, much less intuitive, and takes longer to concoct. The same commentary applies to the second task, below.

Task 2: To compute Sin(x) using its Taylor series expansion

The **PC-1211** implementation is this simple 3-liner:

```
1  "=" AREAD X: Y=X, K=-XX, T=X, N=3, L=E-90
2  T=KT/(NN-N), Y=Y+T, N=N+2: IF ABS TL GOTO 2
3  PRINT Y
```

which is 71 bytes long. The equivalent **HP-41C** program is:

```
01 *LBL "SX"   12 *LBL 01    23  ST+ 01
02  STO 00     13  RCL 02    24  RCL 05
03  STO 01     14  ST* 03    25  RCL 03
04  STO 03     15  RCL 04    26   *
05  X^2        16  X^2       27  ABS
06  CHS        17  RCL 04    28  X#0?
07  STO 02     18   -        29  GTO 01
08   3         19  ST/ 03    30  RCL 01
09  STO 04     20   2        31  END
10  1E-90      21  ST+ 04
11  STO 05     22  RCL 03
```

Sample programs:

These are a few, very short **PC-1211** programs solving simple, well-known problems, for you to see how easily many apparently complex tasks are accomplished in this handheld. A sobering effect would be for you to try and implement them in the **HP-41C**, first without the benefit of previously studying the **PC-1211** solutions, then just trying to mimic them as closely as possible, and see what you come up with in terms of ease of programming and resulting complexity.

Towers of Hanoi

```
1 REM *TOWERS OF HANOI*
2 INPUT "N=";D: A=1,B=3,C=5
3 IF D=1 GOSUB 8: GOTO A(C-1)
4 A(C)=A, A(C+1)=B, A(C+2)=5, B=6-A-B, D=D-1, C=C+3: GOTO 3
5 A=A(C-3), B=A(C-2): GOSUB 8: A(C-1)=6, A=6-A-B: GOTO 3
6 C=C-3, D=D+1: IF C<>5 GOTO A(C-1)
7 BEEP 2: PRINT "DONE": END
8 BEEP 1: PRINT USING "##";"FROM";A;" TO";B: RETURN
```

Hyperbolic functions

```
1 "A" AREAD X: Y=(EXP X-EXP –X)/2: PRINT Y
2 "S" AREAD X: Y=(EXP X+EXP –X)/2: PRINT Y
3 "D" AREAD X: Y=1-2/(EXP 2X+1): PRINT Y
4 "Z" AREAD X: Y=LN(X+V(XX+1: PRINT Y
5 "X" AREAD X: Y=LN(X+V(XX-1: PRINT Y
6 "C" AREAD X: Y=LN((1+X)/(1-X))/2: PRINT Y
```

Compute and display up to 575 decimal digits of *e* = 2.71828...

```
1 CLEAR: INPUT "B=";A: G=5E6, A(7+A)=G, B=2G, F=A
2 FOR D=3 TO 999: C=A(7+F): IF C=0 LET F=F+1, C=A(7+F)
  : IF F=2A GOTO 6
3 FOR E=7+F TO 6+2A: A(E)=INT(C/D,C=A(E+1)+B*(C-DA(E): NEXT E
4 FOR E=6+A TO 7+F-A STEP –1: A(E)=A(E)+A(E+A)
  : IF A(E)>=B LET A(E)=A(E)-B, A(E-1)=A(E-1)+1
5 NEXT E: NEXT D
6 USING "##.#######": BEEP 2: G=G+2B: FOR E=7 TO 6+A
  : PRINT A(E)/B : NEXT E
```

Solve N-Queen puzzle in a general NxN chessboard

```
1 A(Y)=A(Y)+1: IF A(Y)>X LET Y=Y-1: GOTO 1
2 GOTO 5
3 "=" INPUT "N=";X: Y=1
4 A(Y)=1
5 IF Y=1 GOTO 8
6 FOR Z=1 TO Y-1: IF (A(Z)=A(Y))+(Y-Z=ABS(A(Z)-A(Y) LET Z=Y
  : NEXT Z: GOTO 1
7 NEXT Z
8 Y=Y+1: IF Y<=X GOTO 4
9 USING: BEEP 1: FOR W=1 TO X: PRINT "QUEEN AT ";W;A(W)
  : NEXT W: Y=Y-2: GOTO 1
```

What a feeling: Now and Then

When all's said and done, the **HP-41C** upgraded with memory modules first, then with plenty other ROM plug-ins and HP-IL peripherals which were later released, certainly was more powerful and capable than the **PC-1211**, as well as much more expensive. Classic **RPN** language, simple as it is, is more flexible and economic than **BASIC**, if more cryptic and difficult to write, debug and maintain programs using it. **SHARP**'s **BASIC** is much easier to program in than **RPN,** and allows for extremely quick casual program development, the resulting programs being compact, readily understood and modifiable, even months after you wrote them. You don't have to maintain a picture of the stack's contents in your head, nor a map of what's in every numbered register, so it's ideal to concoct quick two-liners without ever using or needing paper and pencil. It is also fully interruptible to perform other computations, then resume program execution, undisturbed.

But at the time, with the **HP-41C** just introduced, no such HP-IL peripherals or ROMs were available, and by the time they were, SHARP had also released new, vastly improved siblings, which were even more difficult to beat. Thus, for its time, the **PC-1211** was a truly worthwhile contender, a top-of-the-line calculator-*cum*-handheld computer unsurpassed by any other model, except arguably the **HP-41C**. It did admit peripherals such as an impact printer and standard audio tapes for program/data storage, and boasted awesome build and design quality, all at a price less than half that of the **HP-41C** alone. Awesome !

Also, you must remember that though these features and programming capabilities in a small handheld aren't surprising to us now in 2006, we're talking about circa 1980 here. The most advanced handheld you could have at the time was the powerful **HP-41C**, then the **SHARP PC-1211** entered the picture. I doubt it made a dent in most highly-committed **HP** fans' awareness, focused as they were in their **RPN**-only world and with their hands all too busy with synthetics, then HP-IL, then even M-code. But for the rest of the world, the **PC-1211** made a profound impact, and ultimately resulted in a decade-long domination of the **BASIC** Pocket Computer scene, with dozens of advanced models, always improving their already very high quality and capabilities. The **HP-75C** and the **HP-71B** were the only **HP** entries in that range, and they barely registered when compared to the **SHARP**s out there. The **SHARP**s sold by the millions in Japan and Europe (most specially Germany and France), and dozens of User Clubs and even publications were created around them. After so many years, and despite the fact that the format has been rendered obsolete by ever small laptops and **PDA**s, they still work as fine as always, remain in top physical shape, and can still be appreciated and put to full use by their proud and loving owners, myself included.
Perhaps you'd consider having a look at them, too, as a welcome complement to your **HP** collection and to expand your views to other horizons. If quality is what attracted you to **HP** in the first place, then classic **SHARP BASIC** handhelds deliver, in spades !.