The PL/I Connection

Issue 3 | IBM Santa Teresa Laboratory | June, 1995

Editor: Christie Nieuwsma

Does IBM Care About PL/I?

I have received many questions and comments regarding IBM's direction and commitment to the PL/I language. Each of you is a valued IBM customer, and we value your loyalty to IBM and PL/I.

IBM is committed to PL/I as a leading enterprise application development environment. While it may appear that other languages are getting more attention, more new technology, and more features, I want to assure you that PL/I is getting significant investment from the IBM Corporation, and has delivered a significant amount of new function over the last two years. We are committed to bringing you the functions that are needed to bring your enterprise into the next century. We are delivering this function on the mainframe as well as on the workstation, meeting your needs for both a robust development environment and a powerful execution environment.

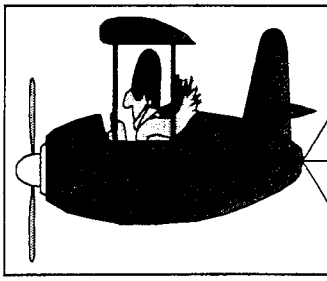
Many of you have asked IBM for new or significantly enhanced PL/I products. The deliveries on MVS, VM, VSE, and OS/2 in the last 12 months are in direct response to your requests. The next product to be shipped is a new PL/I Product for AIX. We are obviously watching the reaction of our customers to these new offerings. The most effective way to ensure that IBM's PL/I investment continues is for you to purchase that which you have asked for.

As we look to the future, we intend to add object-oriented functions to PL/I, as well as provide additional tool, platform and subsystem support. We intend to do this while preserving the fundamental character of PL/I that customers have come to depend on. It is a fundamental part of IBM's AD strategy to leverage our investments across multiple languages, for example, you should expect to see us re-use components from COBOL and C/C++. IBM values your loyalty and commitment to PL/I. We look forward to a continuing partnership with you in the future.

Regards,
John Swainson
IBM Vice President, AD Solutions

Editor's note: John mentioned the products we've delivered over the last 12 months. In case you're unsure of what those are, here's a list. (Note: these are the U.S.A. release dates).

- June 1994 - PL/I for OS/2 Personal Edition
  - 32-bit compiler and debugger
- June 1994 - PL/I for OS/2 Professional Edition
  - 32-bit compiler and debugger
  - Support to access DB2, CICS, IMS C/S, VSAM
- July 1994 - PL/I for OS/2 Toolkit
  - Visual PL/I!
- December 1994 - CODE/370 Support for PL/I
- March 1995 - Language Environment (LE) for MVS & VM Release 4
  - PL/I Multitasking, PLICALLA, PLICALLB
- April 1995 - PL/I for VSE
  - LE/VSE version of PL/I host compiler for VSE



IS THIS YOUR LAST ISSUE?

If you want to continue receiving "The PL/I Connection", we need to hear from you! Please send us the response form at the back of the newsletter. If you are a member of Team PL/I or have previously sent in a response form, you don't need to respond again - you're on our list!

To 'C' or Not to 'C' Is Recoding a Legacy PL/I Application to C The Right Thing to Do?

by Richard Perkinson, Liant Software Corporation
dickp@lpi.liant.com

Editor's note: This is the second article in a series begun in the last (March 1995) issue of "The PL/I Connection."

Part 2 - Recoding From PL/I to C: The General Concerns

There are many things to consider when changing languages, particularly from PL/I to C. What are the costs? What are the risks? How long will it take? What is to be gained or lost? We will consider two levels of concerns for those who are thinking of recoding a PL/I application to C. There are particular language translation issues, as well as more general issues, that stand out from the language specifics.

Code Changes

First and foremost is the obvious fact that the more you have to change, the more costly it will be. When recompiling you may very likely encounter some language variations from one compiler to the next. This could result in changing a few lines in the legacy application to accommodate the new environment. Our experience has been that approximately 1% to 2% of the total number of lines of code need to be changed. This is much less drastic than changing every line of code, as you must when recoding.

Table of Contents

Automatic Code Translators

While it is true that translation tools have been developed to help in converting PL/I to C, the best PL/I to C automatic code translators proclaim up to a 70% successful conversion rate.

My preliminary testing of converters revealed that they are not as satisfying as they might be. Data type conversion and the poor quality of the resulting code were the biggest problems encountered.

Data Type Conversion Problems

All fixed decimal values, regardless of length or precision, were assigned as a Double Float by the translator. By defining all fixed decimals as doubles, all of them occupied 8 bytes of storage rather than the 3, 5, 7, or 9 bytes the files they are in expect them to occupy. This means wholesale data structure conversion must take place before the translated program will run. The original record lengths just don't match the converted structures. On top of this unexpected effort, the loss of precision leads to incorrect results in the runtime routines.

Quality of Converted Code

Concern must also be given to the general quality of the C code generated by a conversion tool. In most cases, this code is not generated in a form that makes for easy reading on the part of programmers doing future maintenance. The program layout, the spacing, indentation, and so forth, tends to get rearranged. If the code is difficult to read and understand, it may hinder later efforts to modernize or re-engineer the recoded application.

Another quality concern of recoding is that because so much of the PL/I logic must be emulated by function calls in C, there is risk of performance deterioration. For example, a series of runtime routines are provided by the translator for a variety of data manipulations for each unsupported data type.

Including the runtime tended to expand the original source listing by a factor between 9 and 17. In addition to performance degradation, the expanded code successfully obscured any sense of the business logic behind each algorithm.

If It Isn't Broken ...

An important fact to remember is that the application is already written in PL/I, and the application works. What new bugs will be introduced into this otherwise working code by just the translation effort itself? As you well know, each line of code that is changed has the potential of introducing a bug that requires time to track down and fix. Pieter Mimno, in the July 1994 issue of *Application Development Trends*, says:

"Tools are available to emulate host-based software such as CICS, VSAM, IMS and DB2 on lower end platforms, to allow legacy applications to run without making a change on the new host processor. The operating environment on the new host processor may be DOS, Windows, OS/2 or Unix. Porting of legacy code to low-cost servers is a proven low-risk, low-cost approach to reducing costs. There are few hidden costs because the ported code is not modified."

Note that Pieter states "because the ported code is not modified". This is an extremely important point. Translation at its essence is a complete modification of the application code. The potential of any change to inadvertently modify the functionality of an otherwise properly functioning program is high.

So, it's not just the time required to make the changes; there is also the time required to fix the problems they cause. The more lines that are changed, the more bugs result. In addition, a certain number of the bugs will surface only after the application goes back into service.

Staff Issues

Converting to C causes other concerns if you have a staff that is very strong in PL/I and not as strong in C. First, if your PL/I staff does the conversion, their C code may not have the quality of their PL/I code. Expect some problems here. Alternatively, if C specialists do the conversion, they may not have a thorough understanding of the PL/I code, and as a result, problems may occur in the translation.

Second, and more important, is the period of productivity loss you will experience. It takes time for a programmer to become adept with a new language. C is not as large a language as PL/I, but it has many irregularities. Once the conversion is complete there will be a period of time when productivity will be somewhat reduced. It is estimated that this lapse in productivity will vary from six to eighteen months. In general, according to The Gartner Group in a recent 1994 survey, training an experienced programmer in a new language takes three projects. If the average project is four to six months, this translates to 12 to 18 months to yield a proficient worker.

According to The Gartner Group, "an internal programmer with multiple language skills", fluent in the new language and familiar with the application being converted, "can do the [recoding] job for around $.50 per line of code. Outside consultants work at a premium, which doubles the cost to $1.00 per line of code".

This article discussed some of the larger, more general issues surrounding the decision to recode from PL/I to C. There are also PL/I specific concerns such as data support, data operations, control structures, file I/O, and built-in functions.

To receive this paper in its entirety, please call 1-800-818-4PLI ext. 221 or (508) 872-8700 ext. 221, or send email to openpl1@lpi.liant.com.

What are they teaching them these days?

Look what students at the University of Northern Iowa can sign up to study:

Topics in Computer Science - Advanced Multi-Threaded Programming.

Topics will include PL/I based OS/2 multi-threaded programming environment, visual development and graphical user interface.

This course will explore the new PL/I language environment for OS/2 especially as it relates to development of graphical user interfaces and advanced

operating system facilities such as multi-threading. PL/I is a language originally introduced in the mid-60's as part of OS/360. It was the implementation language for the Multics operating system and was a model for the development of C, Pascal, and a number of other languages. A dialect is available on VAX/VMS systems. PL/I is a comprehensive language with many facilities not found in C or Pascal. It includes a richer variety of data types, I/O functions, and storage management routines along with a more coherent and readable syntax. The newly implemented OS/2 version is a superset of the mainframe language and contains many facilities to take advantage of the multiprogramming/ multitasking OS/2 environment. It is widely used on mainframe systems to this day, however, there have been no micro-computer implementations until recently.

Why Not Use PL/I?

by Robin Vowels, Royal Melbourne Institute of Technology, Australia
Robin_Vowels@rmit.edu.au

Why you should consider using PL/I:

- It is an easy language to use, the syntax is clean, and the rules are uniform. There are many built-in functions to assist.
- It is a good first language, and provides an excellent growth path for the professional.
- PL/I can be used for solving a wide range of numerical problems in virtually every sphere of the computing arena, be it mathematics, engineering, science, medicine . . . etc.

1. The best part of PL/I is the input and output. You can rely on it.

The input/output is achieved with statements that are ready to use and easy to use. There's no setting up to do. They are *not* procedures or functions dressed up to look like statements.

In formatted output, you don't get silly numbers displayed when you include the format specification, but omit the name of the variable (e.g. C); nor when you use the wrong format (e.g. I4 with a FORTRAN REAL variable). PL/I converts to/from the type that you specify, be it integer, fixed-point, or string. You don't get silly things happening because you left off the "&" from the "scanf" function in C. That's because the input operation (GET) is a statement, not a function.

When you want formatted output, you have a variety of formats that you can choose, such as PICTURE format, with drifting signs and a drifting $ sign. You can insert commas or spaces in very long numbers, to make the output readable (e.g. 123,456,789.356,251 is better than 123456789.356251). You can use either PICTURE formats or the specific codes to get integer, decimal fixed-point, and floating-point outputs. If you don't want to be bothered with formats, you can use simple free-formatted output, e.g. PUT (X);

2. For business applications, you can have decimal data, whereby data is stored exactly (e.g. 5.14 is stored EXACTLY, not as 5.13999), and arithmetic operations give exact results (e.g. 3 x 5.14 gives you 15.42 precisely, not 15.41998).

3. Files are easy to use. If you can do output to the screen, and input from the keyboard, you can do input/output to files, simply by inserting the file name into the GET or PUT statement thus:
GET FILE (MYDATA) etc.

Outstanding and helpful PL/I facilities

1. The debugging facilities are all part of the language (for example, checking for subscript bound errors and string reference errors with SUBSCRIPTRANGE and STRINGRANGE etc. are most helpful, along with appropriate ON statements to display the values of variables).

The best part of this is that you can display everything you need (statement number, values of variables, a trace of procedure calls, etc.) instead of those cryptic messages (from other languages) such as "floating-point trap" at some unidentified location somewhere in the program. (When you've had this message in a 20,000 line program, you'll know what I mean!)

2. If your program gets stuck in a loop, just hit CTRL-BREAK, and your ATTENTION ON-unit will tell you precisely which statement it's executing. Not only that, it lets you do anything you want, including printing the values of all (or preselected) variables, and changing any or all of the values of variables.

You don't have to re-run your program (with new output statements) to find out in which loop it's stuck! (and then run it yet again to display the values of variables you think you need). And if your program was running OK after all, you can resume execution again, as if nothing had happened.

3. You have the ability to intercept errors (via ON statements) and to *continue* execution.

4. You can use the simple, handy I/O statements GET LIST, PUT LIST, along with the very handy PUT DATA statement (for debugging mostly).

5. Everything is "ready to go" unlike some other languages, such as Ada!

6. The "whole-array" operations simplify programming and increase understanding, and increase execution speed into the bargain. If A and B are arrays, you can write A = B; to copy an array. You can use PUT (A); to print an array. (These are trivial examples -- more are available).

7. Dynamic arrays provide the means to write a general program for any size of array. You aren't bound by some artificial limit. You automatically obtain exactly the amount of storage you need for an array.

8. The COMPLEX data type is provided, which simplifies the preparation of programs because common sense complex arithmetic is available. A number of essential functions are at hand for producing complex square root, the conjugate, the absolute value, and for extracting either the real or complex components, and for fabricating a complex number from two real numbers (SQRT, CONJG, ABS, REAL, IMAG, CPLX). The ** operator is available for raising a complex number to a given power. A range of (complex) trigonometric functions is provided (including SIN, COS, SINH, COSH, etc., and mathematical functions such as LOG).

Even more importantly, debugging is simplified, because the error messages are specific to complex number operations. (If you have used complex arithmetic without using the COMPLEX facilities, the best that the error messages can tell you is in terms of real (floating-point) operations, which don't necessarily make sense).

Additionally, the COMPLEX type can be applied to integer as well as floating-point data.

9. PL/I has excellent string-handling facilities. First, there are two types of strings -- the fixed-length string and the varying-length string. These two data types are supported by a range of functions that speed up string processing compared to other languages. It isn't necessary to write your own "string-search" routines like you need to in other languages such as C and Pascal. The search functions are already there (INDEX, SEARCH, SEARCHR). What's more, if the machine has a "search" instruction, it can use that, and search faster than some hand code using a loop. (Having to write your own search routine is like having to turn out your own nuts and bolts on a machine lathe). PL/I can search from either the right-hand end or from the left-hand end of a string (SEARCH and SEARCHR)

Joining strings is a simple operation (concatenation); the length of a string can be discovered by using the LENGTH function; strings containing blanks (or any unwanted character) at the beginning or the end can be removed using the TRIM function. Strings can be centered using the CENTER- built-in functions (helps with page headings etc.) and so on.

Other functions include TRANSLATE, for changing ASCII <==> EBCDIC, and for converting upper to lower case, or for replacing special codes such as TAB to blank or for any other conversion, and VERIFY, for performing a search, validating data, etc.

10. A macro pre-processor is available on IBM mainframe PL/I (OS), IBM PL/I for OS/2, Liant OPEN PL/I, and probably others.

11. And when you need to do something special (rarely, but it's nice to know that you have something to fall back on in an emergency) it's there -- With UNSPEC you can peek at the bits of a data item (be it a character string, an integer, bit string, or floating-point data).

Running CICS PL/I Legacy Programs in 31-bit mode!

by Douglas P. Ewen

If you have legacy PL/I CICS command level programs that must run below the line in your CICS environment, I have the answer for you!

A product that I have developed allows CICS PL/I command level programs to reside and execute above the 16MB line in CICS/MVS, CICS/ESA, and CICS/VSE environments. The name of this product is XAbove/CICS, and it is available now for MVS/XA, MVS/ESA, and VSE/ESA users!

XAbove/CICS is easy to use and it does not require source programs to be changed or recompiled. You must tell XAbove/CICS which PL/I programs it is to handle, and it will do the rest. XAbove/CICS is activated during CICS startup and it insures that the selected CICS PL/I programs are loaded above the 16MB line and execute properly there.

XAbove/CICS is simple to install and implement. You just load the product library onto your system, add CICS entries for the product programs, transactions, and dataset, and away you go! A VSAM dataset must be built that contains the names of the CICS PL/I programs to be handled by XAbove/CICS. An XAbove/CICS transaction allows you to modify this VSAM dataset on-line, so you can add PL/I programs to be loaded above the line dynamically! XAbove/CICS contains a system inquiry transaction that provides summary information on how many PL/I programs are being handled by XAbove/CICS, as well as detailed information on each PL/I program running above the line.

XAbove/CICS optionally allows the PL/I working storage and EXEC CICS GETMAIN storage to reside above the line. You may implement this feature for specific programs or for a set of programs.

XAbove/CICS may be the answer for you if you have legacy PL/I programs that are running below the line in your CICS world! If you would like information on this product, please call Software Pursuits, Inc. at 1-800-367-4823 or (510)769-4900, or fax them at (510)769-4944.

Peter's Performance Tips

by Peter Elderon, IBM PL/I Development
elderon@vnet.ibm.com

In the various releases of OS/2 PL/I, we have introduced a lot of new features, and many of these have received very little publicity. I'd like to discuss here a few that can help improve the performance of your code.

In the June 1994 release, we introduced the compound assignment operators: +=, -=, *=, etc. These can save you typing (it's faster to type, and to read,

```
bills_received_to_date += 1;
```

than

```
bills_received_to_date =
     bills_received_to_date + 1;).
```

Some of them will also improve the performance of your code. Of particular note is the concatenate-and-assign operator ||=.

There are many times when you need to construct a string out of several smaller strings, as in the following example:

```
full_name = path_name || '\' ||
            file_name || '.' || ext;
```

If the target in such an assignment is a varying (or even a varyingz) character string, the compiler will generate much better code if the assignment is broken down into several concatenate-and-assignments, as follows:

```
full_name = path_name;
full_name ||= '\';
full_name ||= file_name;
full_name ||= '.';
full_name ||= ext;
```

The code generated for the series of assignments is better because the compiler uses no temporaries when performing concatenate-and-assigns; it simply assigns the source to the end of the target and updates the length prefix (or moves the null terminator if it is a varyingz string).

It is also worth noting that not only will this code perform faster, but it also requires much less stack space.

Our first OS/2 release contained another new feature, the VALUE attribute, that can also improve the performance of your code. The following code fragment illustrates a situation where you can put it to very profitable use.

```
dcl upper                          char(26)
  init('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
dcl lower                          char(26)
  init('abcdefghijklmnopqrstuvwxyz');
a = translate( b, upper, lower );
```

First, it should be noted that any variable that can be initialized at compile-time, should be declared as STATIC. The default is AUTOMATIC which means that each time the block containing the above code is entered, code will be executed to initialize the two string variables.

However, even declaring these variables as STATIC does not help avoid another problem. Since the second and third arguments in the `translate` are variables, before the translate can be done, code is executed to build the translate table. This can be avoided by changing the attribute INIT above to the attribute VALUE as in the altered form below:

```
dcl upper                          char(26)
 value('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
dcl lower                          char(26)
 value('abcdefghijklmnopqrstuvwxyz');
a = translate( b, upper, lower );
```

The reason that the compiler can now generate better code is that the VALUE attribute declares an item as a named constant, rather than as a variable, and hence the compiler can build the translate table during the compilation.

The VALUE attribute can be used only with scalars and you may not apply the ADDR built-in function to these named constants (or any other kind of PL/I constants). If you have a variable to which these restrictions do not apply, but which you know will not change after it is initialized, it is beneficial to add another new attribute to its declaration - the NONASSIGNABLE attribute.

The NONASSIGNABLE attribute has several advantages:

- It makes your code more self-documenting
- The compiler will flag any (mistaken) attempts to assign to a variable declared as nonassignable
- Nonassignable static variables are placed in the same read-only data segment as all other constants
- If NONASSIGNABLE is specified for a BY-ADDR parameter in an entry declaration, the compiler will not create a temporary if a constant is passed for that parameter

Finally, one of the many new built-in functions introduced in OS/2 PL/I was the REVERSE built-in function. Because of a customer need, in the first CSD this year, the code generated for REVERSE applied to BIT(8) variables is a simple, inline table look-up. However, if the BIT(8) variable is an element of an array, this improved code is generated only if the variable is declared as ALIGNED.

The general lesson here is that any time you declare a BIT variable that is part of an array or structure, if you know the variable is aligned (for instance, when it is the only bit variable in a structure or when all the bit variables in the structure have lengths that are a multiple of 8), you should declare it as ALIGNED.

And, of course, the general lesson from all of the examples above is, once again, the more you can tell the compiler, the better will be the code generated by the compiler. Or, more simply: the nicer you are to the compiler, the nicer it will be to you.

News Flash!

PL/I for OS/2 Professional Edition and PL/I for OS/2 Toolkit are now available at an Independent Software Vendor discount to Commercial/Premier members of the **OS/2 Developer Assistance Program** (DAP). To find out more about the OS/2 DAP in the United States, call (407) 982-6408 or fax (407) 998-7610.

How to Copy Mainframe PL/I Programs to a PC

By Lewis Allen
71662.441@compuserve.com

One of the key features of the PL/I for OS/2 product is that it can compile PL/I source code that was originally written for the mainframe environment. However, to implement a PL/I source program on the PC there are many other things to consider as well as compiling. These include copying the source code to the PC, choosing compiler and linkage options, and setting up the necessary supporting infrastructure such as the user interface, databases, and files.

This article concentrates on the first steps - showing you how to automate the copy of mainframe OS PL/I application source code to the PC OS/2 environment

using a bulk transfer program written in the REXX language.

This article contains examples of REXX code for OS/2 and the mainframe. To run the examples you will need access to an IBM mainframe environment running MVS TSO ISPF that can support the file transfer facility of IBM OS/2 Communications Manager/2. Before continuing, it is probably wise to check that file transfer works in your environment by following these steps:

1. Start OS/2 Communications Manager/2.
2. Logon to your mainframe TSO service using your mainframe User-id and password.
3. Select option 6 from the mainframe TSO/ISPF main menu to display the screen titled 'TSO COMMAND PROCESSOR'. This screen (or a native TSO READY prompt) is required by the file transfer program.
4. Click on the top left button to display the Communication Manager/2 drop down menu.
5. Click on the 'Receive file from Host' option to display the Receive File dialog box.
6. Type in the target workstation file name, the source host file name, and ensure the transfer options are set to 'ASCII' and 'CRLF'.
7. Click on the Receive button to initiate the transfer.

If you encounter problems with file transfer, and you may the first time you try, then these should be resolved before continuing. Communication Manager/2 provides useful help text, you may also be able to get assistance from your local computer support personnel.

Once you have proved that the file transfer works on your PC, you can now start to automate the file transfer. By automation, I mean that we will initiate file transfer from a REXX procedure rather than from the user interface. The sample OS/2 REXX procedure named TEST1.CMD will initiate a file transfer as a background task:

```
/* OS/2 REXX */
/* TEST1.CMD:to fetch PL/I source code*/
   'CALL RECEIVE C:\TEMP\LOBSTER.PLI',
   'LEWIS.LIB.SOURCE(LOBSTER)ASCII CRLF'
```

TEST1.CMD copies the source code of PL/I program LOBSTER from the mainframe library named 'lewis.lib.source' to the OS/2 file named '\TEMP\LOBSTER.PLI' on the c: drive. The file transfer is carried out by the RECEIVE command, which can transfer either ordinary mainframe sequential files or mainframe library members from a partitioned dataset (PDS) to the PC. If your mainframe PL/I source code is held in an alternative proprietary format (such as Endevor or Librarian) then you will need to unload the source code into a partitioned dataset before doing the file transfer.

Try TEST1.CMD on your PC. Type the REXX procedure into an OS/2 editor window, substitute your own mainframe and PC filenames, and save as a file named TEST1.CMD. Ensure that the ISPF 'TSO COMMAND PROCESSOR' screen, or native TSO READY prompt, is displayed in your terminal emulation session (as shown in steps 1 to 3 above) and then run TEST1.CMD from the OS/2 command window by typing the name of the file - TEST1. Once you get this working you can then add a compile step, so that the REXX procedure can both copy and compile, as shown in TEST2.CMD:

```
/* OS/2 REXX */
/* TEST2.CMD : to fetch PL/I source code
 and compile */
  'CALL RECEIVE C:\TEMP\LOBSTER.PLI',
  'LEWIS.LIB.SOURCE(LOBSTER) ASCII CRLF'
  'CALL PLI C:\TEMP\LOBSTER.TXT'
```

Of course, in reality, it would be hardly worthwhile writing a REXX procedure like TEST2.CMD to transfer just one program. But, if we now modify TEST2.CMD to make it more generic so that it will accept any program name and then add a loop so that it will repeat for a set of programs, it is transformed into a useful bulk transfer tool. For example, if you have a mainframe application of 1000 programs to copy to the PC, you could start the bulk transfer going as a background task, and get on with some other work while your entire mainframe application is transferred to OS/2.

The changes to the REXX procedure to support a bulk transfer are fairly straightforward. We need to add the ability to drive the REXX from a PARTS file containing the names of all the libraries and programs to be transferred from the mainframe, and we need to add the ability to LOG the return code from each compile, so that when the task has finished, we can look in a log file to see which compiles have worked and which have failed. The improved REXX procedure, named XFEROS2.CMD will do this:

```
/* OS/2 REXX */
/* XFEROS2.CMD : to bulk fetch PL/I
   source code and compile */
```

```
/* trace i */
parts = 'c:\temp\PARTS.TXT' /* name of
                              parts file */
log = 'c:\temp\LOG.TXT' /* name of log
                                  file */
pcdrive = 'C'          /* default drive */
pcpath  = '\temp'       /* default path */
do while lines(parts) > 0
  source = linein(parts)
  if source = ' ' then leave
parse var source type dsn member therest
hostfile = dsn'('member')'
pcfile = pcdrive':'pcpath'\'member'.PLI'
'CALL RECEIVE' pcfile hostfile 'ASCII
      CRLF'
call  lineout  log,  time()  'TRANSFER'
pcfile copies('*',rc)
if type = 'PLI' then
 do
   'CALL PLI' pcfile '(GONUMBER)'
   call lineout log, time() 'COMPILE '
      pcfile copies('*',rc)
 end
end
return
```

The parts file used by this procedure simply contains a line for each library and program to be transferred from the mainframe. It might look like this:

```
INC lewis.lib.source LOB001I
PLI lewis.lib.source LOB001E
PLI lewis.lib.source LOB002E
PLI lewis.lib.source LOBnnnE
```

The parts file can either be created manually using a PC editor, or if there are a large number of programs, it can be created by a program on the mainframe and then downloaded to the PC before running the bulk transfer. The mainframe REXX procedure named XFERMVS shows one way of producing a parts file on the mainframe:

```
/* TSO ISPF REXX */
/* XFERMVS : To generate a mainframe
      parts file */

number_of_libraries = 1
library.1 = 'lewis.lib.source' /* main
                  frame source library */
parts = 'lewis.lib.parts' /* parts file
                        to be created */
call open
do i = 1 to number_of_libraries
 "ISPEXEC LMINIT DATAID(DATAID)
    DATASET('"library.i"')"
 "ISPEXEC LMOPEN DATAID("dataid")
    OPTION(INPUT)"
 do while rc = 0          /* process each
                                  member */
   "ISPEXEC LMMLIST DATAID("dataid")",
     "OPTION(LIST) MEMBER(MEMBER)
      STATS(NO)"
   if rc <> 0 then leave
   call write 'PLI' library.i member
 end
 "ISPEXEC LMCLOSE DATAID("dataid")"
end
call close
return

open:procedure expose parts
 "FREE DD(PARTS)"
 if sysdsn("'"parts"'") = 'OK' then
  do
   "ALLOC DD(PARTS) DSN('"parts"') OLD"
  end
  else do
   "ALLOC DD(PARTS) DSN('"parts"') ",
     "NEW CATALOG RECFM(F B) LRECL(80)",
     "SPACE(10,10) TRACKS DSORG(PS)
     RELEASE"
  end
return

write:procedure
 arg output
 oput.1 = output
 "execio 1 diskw PARTS (stem oput.)"
return

close:procedure
 "execio 0 diskw PARTS (finis)"
 "FREE DD(PARTS)"
return
```

To get the REXX procedure XFERMVS to work, you first need to copy it into a REXX library in the MVS TSO ISPF environment, then edit it to replace 'lewis.lib.source' with the name of your source library, and replace 'lewis.lib.parts' with the name of the parts file you wish to create. Be careful here since XFERMVS will overwrite any parts file dataset previously created. Run XFERMVS by typing 'TSO XFERMVS' on any ISPF command line.

The XFERMVS procedure shown here is intentionally minimal but provides a framework that you can tailor to your own mainframe environment. For example, it could be made more flexible by adding a user interface so that a user could enter the required source library names, and range of member names, perhaps supporting wild cards. Another enhancement could add the function to transfer and process other objects such as DB2 definitions, data files, and so on.

To summarize, the method to bulk transfer programs from mainframe to PC is:

1. Put the source programs you want to download into a mainframe library partitioned dataset (PDS).

2. Run the mainframe REXX procedure XFERMVS against the PDS(s) to produce a parts file.
3. Copy the parts file down to the PC.
4. Run the OS/2 REXX procedure XFEROS2 to download and compile each program in the parts file and produce a log file of results.

If you need help with writing OS/2 REXX, try the on-line OS/2 REXX INFORMATION help. I also recommend *'A Practical Approach to programming the REXX Language'* by M.F.Cowlishaw, Prentice Hall ISBN 0-13-780651-5. For help with MVS REXX, try *'TSO Extensions Version 2 Procedures Language MVS/REXX Reference'*, IBM publication number SC28-1883, and *'TSO Extensions Version 2 Procedures Language MVS/REXX User's Guide'*, SC28-1882. For help with ISPF services try *'ISPF Dialog Management Guide'*, SC34-4112, and *'ISPF Dialog Management Services and Examples'*, SC34-4113. For help with the ISPF Library Management services look at the examples at the back of *'ISPF MVS Services'*, SC34-4023.

Lewis Allen is an independent IT consultant based in London, England specializing in IBM PL/I, CICS and DB2.

Upcoming Tradeshows

There are several tradeshows over the next few months at which we intend to talk about or demo PL/I. We'd love to have you come by and see first hand what we're up to these days.

GUIDE	July 16 - 20 in Boston
OS/2 World	July 18 - 21 in Boston
Enterprise Computing	July 26 - 28 in Chicago
SHARE	August 13 - 18 in Orlando
Networld/Interop	September 25 - 29 in Atlanta

Fax Information Service

Did you know that IBM has a "Fax on Demand" service that allows you to easily retrieve documents for immediate delivery to any fax machine? There is information available on all sorts of IBM and IBM Business Partner products, including PL/I. The service is available in the USA 24 hours a day, 7 days a week. Voice prompts navigate you through the document selection process. To use the IBM Fax service, call:

1-800-IBM-4FAX (1-800-426-4329).

Probably the best way to start is to order the index, and then pick out specific documents from there.

New PL/I for OS/2 Redbook

A new IBM Redbook, titled "*PL/I for OS/2; PL/I for OS/2 Toolkit: Visual PL/I; CODE/370 PL/I Support*" is now available. It's publication number is: GG24-2501-00. The abstract follows:

"This document is unique in its detailed coverage of PL/I for OS/2 and Visual PL/I provided with the PL/I for OS/2 Toolkit. It focuses on the use of these products for development of MVS host applications or OS/2 applications with a graphical user interface. It provides information about interfacing PL/I for OS/2 with DB2 and CICS and using CODE/370 to develop and test PL/I host applications on the workstation.

This document is written for PL/I application designers and programmers who are moving their development environment from the MVS host to OS/2 workstations. Knowledge of the PL/I language and OS/2 workstations is assumed."

If you have any questions about Redbooks, you can send a note to redbook@vnet.ibm.com.

Syntax Highlighting for PL/I

by David W. Noon, Australian-at-large

What? Why?

What is syntax highlighting? It is an approach to editing program source code that allows the programmer to distinguish language keywords from program variables and constants - at a glance.

Why use it? Syntax highlighting allows a programmer to see potential conflicts between usage of a name as a program variable and a language keyword, to find language keywords that can cause a reported problem and, most simply, to provide a visual structure to each statement in the source code.

The Early Days

In years gone by, color screens were expensive and most programmers were limited to using monochrome displays. Since most programming languages are case insensitive, a rudimentary form of syntax highlighting arose. This was to use all capital letters for language keywords and mixed case for program variables. Since this newsletter is in black and white, a simple example in PL/I will illustrate this nicely.

```
My_prog:
PROC OPTIONS(MAIN REENTRANT) REORDER;
DCL Some_string     CHAR(30) VAR,
    Max_page_size   BIN FIXED(31,0)
                    STATIC INIT(60),
    SYSIN           STREAM INPUT,
    SYSPRINT        PRINT;

    OPEN FILE(SYSIN),FILE(SYSPRINT)
       PAGESIZE(Max_page_size);
    GET FILE(SYSIN) LIST(Some_string);
    PUT FILE(SYSPRINT) EDIT
   ('The string read was: "',Some_string.'"')
    (A);
    CLOSE FILE(SYSIN),
    FILE(SYSPRINT);
END My_prog;
```

As you can see, the program variables are quite distinct in their appearance from the PL/I language keywords. I have left the names of the standard files, SYSIN and SYSPRINT, in all capital letters, since these are more or less part of the language, even though they can be redeclared.

Others take a reverse approach and use all lower case letters for language keywords, but since the mixed case variable names tend to be predominantly lower case, this tends to be less distinct than all capitals for keywords.

The main points are:

- You can choose your own style
- The program listing is much more easily read than in the days when all the source code was punched on IBM 029 keypunch machines.

In Living Color

These days, nearly all of us have color screens attached to high-resolution video adapters. This does, of course, add a new dimension to the display of program source code. In addition to lexical distinction with letter cases, we can now use different colors to distinguish elements of the source code. This allows us to classify language keywords into different types, to distinguish constants from variables, and to classify constants into different types.

By choosing different colors for declarative keywords and executable keywords, we can readily identify the type of statement we are looking at, without having to examine its syntax in detail. In particular, long runs of executable statements, interrupted by declarative statements such as those for a BEGIN block, can show up the hierarchical ownership of PL/I s automatic storage. This can be seen very readily by scrolling your screen and looking for the change in color.

Similarly, mismatches between source and target can often be seen when different types of constants are highlighted using different colors. Such things as assigning a floating point number to a character string, which PL/I permits but which can cause unexpected results, can be seen at a glance when floating point numbers have a different color from character strings. This is most effective when program variables have names that reflect their type.[1]

At present, we can use version 6.0 of the OS/2 Enhanced Editor (EPM.EXE) to recognize the language elements and assign colors to them. This is done by means of a text file that defines the colors - the editor uses a palette of 16 - for each classification, and the language keywords and punctuation that the editor can use for its recognition of the language syntax. You can

[1] Charles Szimonyi of Microsoft has written at length on his 'Hungarian' naming style, which would be effective in this situation. However, this is somewhat tangential to the topic at hand.

obtain a copy of a PL/I keywords file, called EPMKWDS.PLI, by downloading EPMPLI.ZIP from CompuServe s OS2DF1 forum or from the OS/2 Shareware BBS at +1-703-385-4325. It might also be available from other FidoNet bulletin boards. To invoke PL/I syntax highlighting, ensure that this file is somewhere in the EPMPATH directories, open the EPM command dialogue and issue the command:

```
TOGGLE_PARSE 1 EPMKWDS.PLI
```

While this approach of recognizing keywords without regard to context is very simple, it is not completely satisfactory.

Oooh! Such Language!

The very flexibility of PL/I, while solving a multitude of programming problems, presents a problem for syntax highlighting. In particular, PL/I uses some of its language keywords for slightly different purposes in different contexts. For example, the word BINARY is used to declare a variable that is to contain a number (fixed or floating point) represented in binary, but it is also the name of a built-in function that converts a value from some other representation into binary. This poses the editor with the problem of resolving this ambiguity: does it highlight it as a declarative keyword or as a built-in function reference?

The only real solution is to create a text editor that can parse source code text against a definition of the PL/I grammar. A number of compiler vendors, most notably Borland, have done this for other language compilers by incorporating the compiler into an Integrated Development Environment, or IDE. This has created a strong relationship between the text editor and the compiler. Anybody who has used the Borland IDE editor to edit files other than program source will be aware that this approach has its limitations too. It keeps coloring until the editor works out that the text is not C/C++ or Pascal source, usually when you save the file. In such a case, the syntax highlighting is usually a distraction.

The ideal solution is an editor that has an extension facility to read a grammar definition, including ways to resolve context-sensitive ambiguity. With such a tool, the compiler vendors need not create a custom editor for each language, but merely a grammar definition for the *parsing editor.*

Until such an editor exists - either associated with the compiler or an abstract parsing editor - we PL/I luminaries will have to be content with keyword recognition by EPM.

(Editor's note: IBM has recently supplied PL/I Language Sensitive Editing as part of the CODE/370 product. We have also announced our intention to provide Language Sensitive Editing support for the PL/I for OS/2 products based on the well-received LPEX editor. If you are interested in participating in a limited minibeta for this LPEX support, please contact Karen Barney, krbarney@vnet.ibm.com. You can also contact us using the addresses given in the "What Next?" section at the end of the newsletter).

PL/I Services Survey

To enable IBM's new, expanded PL/I capabilities quickly and with a minimum of effort, we are considering offering services focused on important subject areas like client/server that can extend your ability to use PL/I effectively in every part of your enterprise. Please note that these services are currently *NOT AVAILABLE* and references here do not imply that IBM will make these available in the future.

The following are possible services that we are considering:

1. **Quickstart** - Getting you up and running quickly.

   Designed to quickly install and configure the development environment for IBM PL/I for OS/2, demonstrating the functional capability and integration of IBM PL/I into your workstation environment.

2. **Proof of Concept** - Showing you how it works in your environment.

   Based on special requirements, IBM consultants demonstrate the value of IBM PL/I's technology through a customer selected application. This is a powerful way to get specific information about how the new IBM PL/I technology applies to your current environment.

   With this service, you eliminate the need to learn how to use new technology before you fully evaluate the

technology and take advantage of it.

3. **Migration** - Moving your applications to take advantage of new technology.

   Ensures a smooth migration from an IBM legacy PL/I environment to one taking advantage of features found in the new technology found in the new IBM PL/I family of products. This service may also assess applications migrating from a mainframe to a workstation environment.

4. **AD Environment Assessment** - Gaining insight into your options.

   If you are uncertain about where to start implementing the new IBM PL/I technology in your environment, this service provides insights into your company's current and emerging environment and application portfolio. The consultants evaluate your IT infra-structure and note critical success factors, as well as identify pilot applications for development. The consultants' final report includes advice on the skills you need to develop, the best runtime configuration to target, a viable technology road map to follow, and activities to prioritize.

We would appreciate your input on the value of these services and suggestions on services we should be developing. Please send your comments and suggestions to Wilbert Kho via one of the following:

1. PLI Forum on IBM TalkLink
2. IBM Mail: USIB5DWS at IBMMAIL
3. Internet: wilbert@vnet.ibm.com
4. Mail: Wilbert Kho
   IBM Santa Teresa Lab
   L05/H252
   P.O. Box 49023
   San Jose, CA 95161-9023
   U.S.A.

What Next?

To continue receiving "The PL/I Connection" newsletter, send your response to one of the following addresses. If you're a member of Team PL/I or have already responded, you don't need to do so again.

PL/I Newsletter
IBM - Santa Teresa Lab.
555 Bailey Avenue, J84/D245
P.O. Box 49023
San Jose, CA 95161-9023
U.S.A.

Fax (408) 463-4820

teampli@vnet.ibm.com
USIB5RLG at IBMMAIL

Questions and Answers

Question	Answer
I have lost my desktop completely and, what's worse, I do not have a backup copy of the .INI or .RC files to MAKEINI! I would really like to have my PL/I for OS/2 and PL/I Toolkit folders back...	If your folders and icons are gone, you should be able to get them back by using Diskette #1 of the original product and typing INSTALL. Choose "Update the currently installed product" and "Continue". You will then be told that your product is at the most recent level and asked if you wish to continue. Reply "YES" and the Install will buzz through looking for anything that was not up to date, which in your case will be the folder and the icons.
I would like to call a non-PL/I (C) routine that produces a 32 bit return value. PLIRETV only gives me the lower 16 bits. Is there a way to obtain the full 32 bit return value?	PLIRETV was extended to 32 bit in PL/I for MVS & VM. You need to compile your PL/I routine with PL/I for MVS & VM in order to take advantage of it.
With the PL/I for OS/2 Toolkit, how does one back up a project so that it can be completely restored for subsequent changes or moved to another machine? Which files are vital and how is the desktop restored?	To back up a Visual PL/I project, save the .PMG file for the project. Also, if you have created any "My Code Blocks" which are used by the project, you should back up PMGMYCB.PLF too. (Don't worry about "Any Code" blocks - they go into the .PMG file). To restore the project on a new system, right click in the main Visual PL/I window and select the "Load Project" menu item. You will then be asked to specify the .PMG file you want loaded. The project will then appear in the main Visual PL/I window.
I am trying to call a C/370 subroutine from PL/I. C expects the actual value of an argument, but PL/I passes a pointer to a field containing the value. How can I get PL/I to pass the actual value?	The BYVALUE attribute in the PL/I for MVS & VM product will solve your problem. **Note:** BYVALUE is also available in PL/I for OS/2.
I am trying to install the CSD for PL/I for OS/2. I get a message saying I don't have the compiler installed. Any ideas what's happening?	Software Installer (the installation program used by PL/I for OS/2) saves install history information in \OS2\SYSTEM\EPFIS.INI on your boot drive. If you've deleted this file, if it's been corrupted, or if you're booting from a different drive than when you installed PL/I for OS/2, Software Installer will think that PL/I has not been installed, and will refuse to install the CSD. You'll probably need to reinstall PL/I from the original diskettes, and then install the CSD.
I'm starting to prepare for the year 2000 change. Does PL/I support dates in the year 2000 and beyond? If it does, how is this achieved?	PL/I 2.3 and PL/I for OS/2 have the Built-In Function, DATETIME, which returns a four digit year in the format: yyyymmddhhmmssttt There are also several date/time callable services in LE/370 that you can take advantage of with PL/I for MVS & VM that can help you with this.

COPYRIGHT NOTICE

Trademarks

The information in this document concerning non-IBM products was obtained from the suppliers of those products or from their published announcements. IBM has not tested those products and cannot confirm the accuracy of the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products

The PL/I Connection Response Form

Yes, I would like to continue receiving the PL/I Newsletter.

Name:

Company:

Address:

Phone Number:

Comments: