IBM XL Fortran for AIX

Discover Uninitialized Floating point and Integer variables

By: Jim Xia

Level: Introductory

May 2012

Contents

IBM XL Fortran for AIX........ 1

About this Tutorial 4

Objectives 4

Prerequisites 4

System Requirements........ 4

Glossary 5

Start the Terminal Emulator to AIX System........ 6

Get Started with IBM XL Fortran Compiler 6

Conclusion 15

Trademarks........ 15

Resources........ 15

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Before you start

About this Tutorial

This demo demonstrates the use the -qinitauto= option of IBM XL Fortran compiler for debugging errors due to uninitialized variables in Fortran programs. The principles demonstrated here also apply to C and C++ programs using the IBM XL C/C++ compilers.

These options are useful while moving applications from older platforms or other vendors to the IBM XL Fortran or IBM XL C/C++ compilers. A common problem encountered during migration is that an old program that works flawlessly, fails to produce expected results in the new environment or suddenly due to a unrelated change. Often this is caused by common bug of using uninitialized variables in the program. For example, uninitialized variables may accidently contain values that are insignificant to the program execution, hindering their discovery at development stage. The demo here is to show an easy-to-apply strategy that leads to discovering of these variables.

Objectives

- Use IBM XL Fortran Compiler for AIX to compile Fortran source code with "–qinitauto" option to discover uninitialized variables

- Total time: 30 minutes

Prerequisites

- Basic Unix skills
- Basic Source code compile and build experience

System Requirements

http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/sysreq

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Glossary

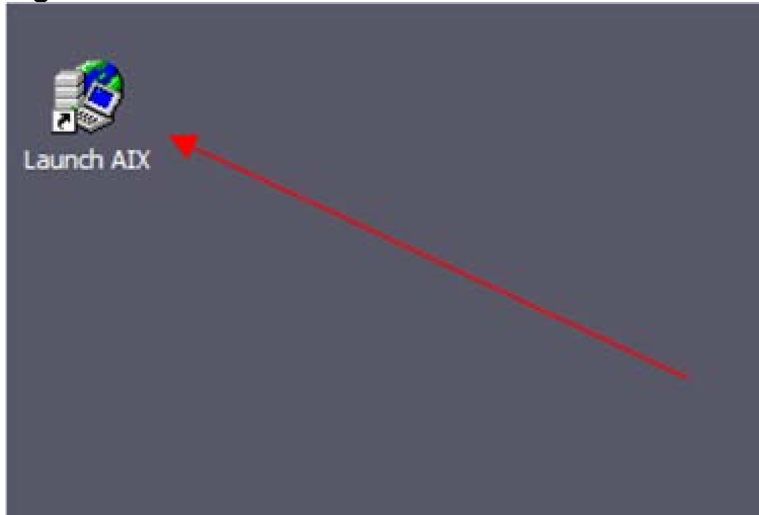
IBM XL Fortran Compiler: The IBM® XL Fortran compiler offers advanced compiler and optimization technologies and is built on a common code base for easier porting of your applications between platforms. It complies with the latest Fortran international standards and industry specifications and supports a large array of common language features.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Getting Started

Start the Terminal Emulator to AIX System

Figure 1 Get Started

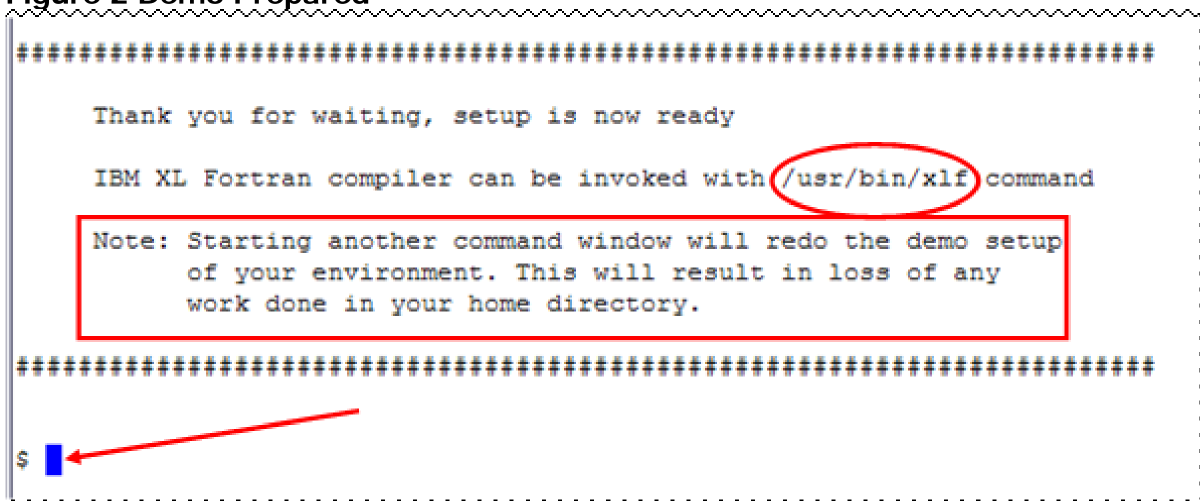


Double click the "Launch AIX" icon on the desktop (See Figure 1) to start the character terminal to AIX system.

Get Started with IBM XL Fortran Compiler

A successful login will result with the user presented with a menu of the demos hosted on the server. Type 2 and press Enter to select the "XL Fortran detecting uninitialized variables" demo.

Figure 2 Demo Prepared



Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

On the terminal window you will see important information and the directory path to the compiler invocation command (See Figure 2 Demo Prepared *oval red*).

Note: Starting another command window will start the demonstration setup of your environment. This will result in loss of any work done in your home directory. This will impact any progress you have made on demo steps going forward.

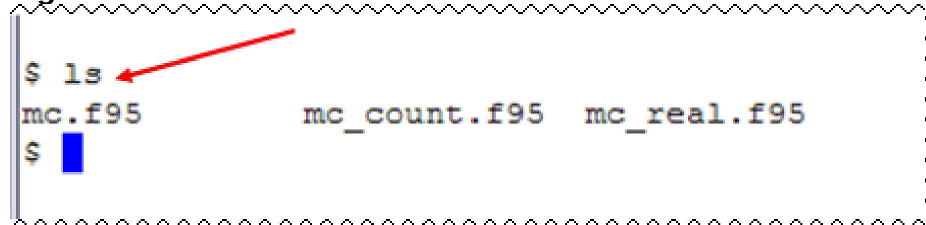
This demo does not require more than one terminal window. However, if you prefer more than one terminal window then you may open them before going forward.

The terminal window is now ready for commands (See Figure 2 Demo Prepared *arrow*). Your home directory contains the source code for three Fortran programs. Type the ls command to see the directory content (See Figure 3 Contents).

Command:

```
ls
```

Figure 3 Contents



The Fortran programs are written in Fortran 95. They use a Monte Carlo simulation method to compute the constant value of Pi (3.1415926...). At every 1000 points generation, a computed Pi value is printed on the screen, and then the program asks if the user wants to run more simulations. Note it is not true that the more runs you have, the more accurate the value will be. This is due to computational errors accumulated through each iteration. Users are also given a choice to provide their own random number seeds when doing the simulation so that the simulations does not depend~~s~~ on the default seed setting given by the compiler.

Three F95 programs are provided here:

mc.f95	The correct version of Monte Carlo simulation
mc_real.f95	Same as mc.f95 except two float variables are NOT initialized
mc_count.f95	Same as mc.f95 except two integer variables are NOT initialized

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Steps:

1. Compile and run mc.f95 program. See figure 4

   Command:
   ```
   xlf -o mc mc.f95
   ```

   Figure 4 Compile mc.f95

   ```
   $
   $ xlf -o mc mc.f95
   ** m_point   === End of Compilation 1 ===
   ** montecarlopi   === End of Compilation 2 ===
   1501-510  Compilation successful for file mc.f95.
   $
   ```

   Run mc program (See Figure 5 Run mc).

   Command:
   ```
   mc
   ```

   Figure 5 Run Configure

   ```
   $
   $ mc
   type in  2 random integer seed values:
   ```

   Enter two numbers and iterate to calculate Pi value (See Figure 6 Enter number)

   Type a number, press spacebar and then type another number. Press enter to run the program further. Pi value will be computed on the first iteration. Program will stop for users input after each iteration. Type y and then press enter to execute next iteration. Continue the iterations until value of Pi approached 3.14. Press n to discontinue the program. (See Figure 6)

   Figure 6 Enter number

   ```
   $
   $ mc
   type in  2 random integer seed values: 10 20
   After     1 runs, the computed pi value =     3.09600
   Continue? (Y/N) y
   After     2 runs, the computed pi value =     3.11400
   Continue? (Y/N) y
   After     3 runs, the computed pi value =     3.14133
   Continue? (Y/N) n
   $
   ```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

The above test is to demonstrate the correct execution flow of the program written to calculate the value of Pi using the Monte Carlo simulation method.

2. Compile and run mc_real.f95. See figure 7 compile mc_real.f95

Note: mc_real.f95 has two uninitialized float variables, total_hit and miss. Look at the program at line 43 and 44 where the initialization statements are commented.

Program mc_real.f95

```
…
…
 43  !    total_hit = 0.0
 44  !    miss = 0.0
…
…
```

(Note "!" in Fortran is comments rest for the line similar to "//" in C++)

The computation using these two variables is at lines 62 and 63 of the mc_real.f95 program

Program mc_real.f95

```
…
…
 62          total_hit = total_hit + in
 63          miss = miss + out
…
…
```

Later total_hit and miss variables are used in computing the value of pi on line 68 of the program mc_real.95

Program mc_real.f95

```
…
…
68          answer = total_hit/(total_hit + miss)
…
…
```

Compile mc_real.f95 program.

Command:

```
xlf –o mc_real mc_real.f95
```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure 7 compile mc_real.f95

```
$
$ xlf -o mc_real mc_real.f95
** m_point   === End of Compilation 1 ===
** montecarlopi   === End of Compilation 2 ===
1501-510  Compilation successful for file mc_real.f95.
$
```

Run mc_real program (See Figure 8 Run mc_real).

Command:
```
mc_real
```

Enter two numbers and iterate over Pi value calculation as shown in figure 8 Run mc_real

Type a number, press spacebar and then type another number. Press enter to run the program further. Pi value will be computed on the first iteration. Program will stop for users input after each iteration. Type y and then press enter to execute next iteration. Continue the iterations until value of Pi approached 3.14. Press n to discontinue the program. See Figure 8 Run mc_real

Figure 8 Run mc_real

```
$
$ mc_real
type in  2 random integer seed values: 10 20
After    1 runs, the computed pi value =     3.09600
Continue? (Y/N) y
After    2 runs, the computed pi value =     3.11400
Continue? (Y/N) y
After    3 runs, the computed pi value =     3.14133
Continue? (Y/N) n
$
```

Note: The program seems to work correctly even with uninitialized variables. This gives an illusion that the program is defect free.

3. Compile using -qinitauto=ff compiler option and run mc_real.f95 program

Command:
```
xlf -o mc_real1 -qinitauto=ff mc_real.f95
```

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Figure 9 Compile mc_real with –qinitauto=ff

```
$
$ xlf -o mc_real1 -qinitauto=ff mc_real.f95
** m_point    === End of Compilation 1 ===
** montecarlopi    === End of Compilation 2 ===
1501-510  Compilation successful for file mc_real.f95.
$
```

The Compiler option –qinitauto=ff, initializes all of the uninitialized automatic real variables variables to -NaN(Q).

Run mc_real program (See Figure 10 Run mc_real with –qinitauto=ff).

Command:

```
mc_real1
```

Enter two numbers and iterate over Pi value calculation as shown in figure 10 Run mc_real with –qinitauto=ff

Type a number, press spacebar and then type another number. Press enter to run the program further. Pi value will be computed to NaN (not a number) value on the first iteration. Program will stop for users input after each iteration. Type y and then press enter to execute next iteration. Continue up to 3 iterations. Press n to discontinue the program. See Figure 10 Run mc_real with –qinitauto=ff

Figure 10 Run mc_real with –qinitauto=ff

```
$
$ mc_real1
type in  2 random integer seed values: 10 20
After     1 runs, the computed pi value =          -NaNQ
Continue? (Y/N)  y
After     2 runs, the computed pi value =          -NaNQ
Continue? (Y/N)  y
After     3 runs, the computed pi value =          -NaNQ
Continue? (Y/N)  n
$
```

Note: That values given in the red box are not right result and something is wrong, computed value is always -NaN(Q)

The resulting NaNs (when -qinitauto=ff turned on) in the computations that are used to produce correct results normally indicate uninitialized floating point variables. You may try on your own to recompile the program with -qinitauto=ff flag after uncommenting the initialization statements at lines 43 and 44 in mc_real.f95 program to see if that is the fix for the NaN issue.

4. Compile the mc_count.f95 program using the -qinitauto=ff compiler option and run the mc_count program

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Note: mc_count.f95 has two uninitialized integer variables, in and out. Look at the program at lines 51-52 where the initialization statements are commented.

Program mc_count.f95

```
…
…
 51   !          in = 0
 52   !          out = 0
…
…
```

(Note "!" in Fortran is comments rest for the line similar to "//" in C++)

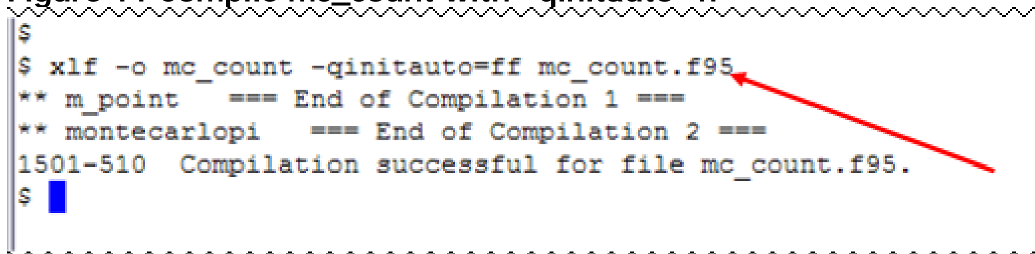
The in and out variables are used to sum up the "hits" and "misses" every 1000 random points used in the Monte Carlo simulation at lines 54 though 63 of mc_count.f95.

Program mc_count.f95

```
…
…
 54           do i = 1, size(darts)
 55               if ((darts(i)%x**2 + darts(i)%y**2) < 1.0_rkind) then
 56                   in = in + 1
 57               else
 58                   out = out + 1
 59               end if
 60           end do
 61
 62           total_hit = total_hit + in
 63           miss = miss + out
…
…
```

Command:
```
xlf –o mc_count -qinitauto=ff mc_count.f95
```

Figure 11 Compile mc_count with –qinitauto=ff



```
$
$ xlf -o mc_count -qinitauto=ff mc_count.f95
** m_point   === End of Compilation 1 ===
** montecarlopi   === End of Compilation 2 ===
1501-510  Compilation successful for file mc_count.f95.
$
```

Run mc_count program (See Figure 12 Run mc_count with –qinitauto=ff).

Command:
```
mc_count
```

Enter two numbers and iterate over Pi value calculation as shown in figure 12 Run mc_count with –qinitauto=ff

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Type a number, press the spacebar and then type another number. Press enter to run the program further. Pi value will be computed on the first iteration. Program will stop for users input after each iteration. Type y and then press enter to execute next iteration. Continue the iterations until value of Pi approached 3.14. Press n to discontinue the program. See Figure 12 Run mc_count with –qinitauto=ff

Figure 12 Run mc_count with –qinitauto=ff

```
$
$ mc_count
type in  2 random integer seed values: 10 20
After     1 runs, the computed pi value =     3.09820
Continue? (Y/N) y
After     2 runs, the computed pi value =     3.10948
Continue? (Y/N) y
After     3 runs, the computed pi value =     3.12579
Continue? (Y/N) y
After     4 runs, the computed pi value =     3.13491
Continue? (Y/N) y
After     5 runs, the computed pi value =     3.14877
Continue? (Y/N) n
$
```

Note: That program seems to be running correctly even after it took more iterations for Pi value to approach 3.14. This may seem normal to users who have not seen the examples of this program given in the previous steps.

5. Compile mc_count.f95 using -qinitauto=0f compiler option and run mc_count1 program.

   Note: Option –qinitauto=0f will initialize all 32-bit integers as 252645135 (bit pattern: 0x0f0f0f0f).

   Command:
   ```
   xlf -o mc_count -qinitauto=0f mc_count.f95
   ```

Figure 13 Compile mc_count with –qinitauto=0f

```
$
$ xlf -o mc_count1 -qinitauto=0f mc_count.f95
** m_point    === End of Compilation 1 ===
** montecarlopi    === End of Compilation 2 ===
1501-510  Compilation successful for file mc_count.f95.
$
```

Run mc_count1 program (See Figure 14 Run mc_count1 with –qinitauto=0f).

Command:
```
mc_count1
```

Enter two numbers and iterate over Pi value calculation as shown in figure 14 Run mc_count with –qinitauto=0f

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

Type a number, press spacebar and then type another number. Press enter to run the program further. Pi value will be computed on the first iteration. Program will stop for users input after each iteration. Type y and then press enter to execute next iteration. Continue the at least 5 iterations. Press n to discontinue the program. See Figure 14 Run mc_count with –qinitauto=0f

Figure 14 Run mc_count with –qinitauto=0f

```
$
$ mc_count1
type in  2 random integer seed values: 10 20
After     1 runs, the computed pi value =     2.00000
Continue? (Y/N)  y
After     2 runs, the computed pi value =     2.00000
Continue? (Y/N)  y
After     3 runs, the computed pi value =     2.00000
Continue? (Y/N)  y
After     4 runs, the computed pi value =     2.00000
Continue? (Y/N)  y
After     5 runs, the computed pi value =     2.00000
Continue? (Y/N)  n
$
```

Note: The value of Pi is always 2.00000 through the iterations. This strange behavior of the program should immediately get noticed by the user.

The problem shown here is due to the uninitialized variables in and out at lines 51 and 52 of the program mc_count.f95. These kind of errors are common in real applications

The statements at lines 51 and 52 re-initialize the counters on every iteration.

Program mc_count.f95

```
…
…
 51   !         in = 0
 52   !         out = 0
…
…
```

Since these variables are of the type integer, -qinitauto=ff will initialize them to -1, the calculation variations is too little to detect the bug. However, with -qinitauto=0f, they're initialized as a big positive number, thus magnifying the impact.

You may try on your own to recompile the program after uncommenting the initialization statements at lines 51 and 52 in mc_count.f95 program to see if that fixes the issue of repetitive Pi value 2.00000 using the –qinitauto=0f compiler option.

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.

What you have learned

In this exercise you learned how to:

- Use the IBM XL Fortran for AIX compiler to compile programs.
- Use –qinitauto flag to set default value for float and integer variables.
- -qinitauto=ff is effective to discover if your program has used uninitialized floating points variables.
- -qinitauto=0f is often effective to see if your program has used uninitialized integers.

Conclusion

This concludes the tutorial on using the IBM XL Fortran compiler. This tutorial has shown how to compile programs with the IBM Fortran for AIX compiler and use the -qinitauto option to initialize uninitialized automatic variables to a specific value.

Trademarks

IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product and service names may be trademarks or service marks of others.

Resources

XL Fortran for AIX, Library :

http://www.ibm.com/software/awdtools/fortran/xlfortran/aix/library/

Community Cafe :

http://www.ibm.com/software/rational/cafe/community/ccpp

Copyright © 2012, IBM® Corporation. Published by IBM® developerWorks®.