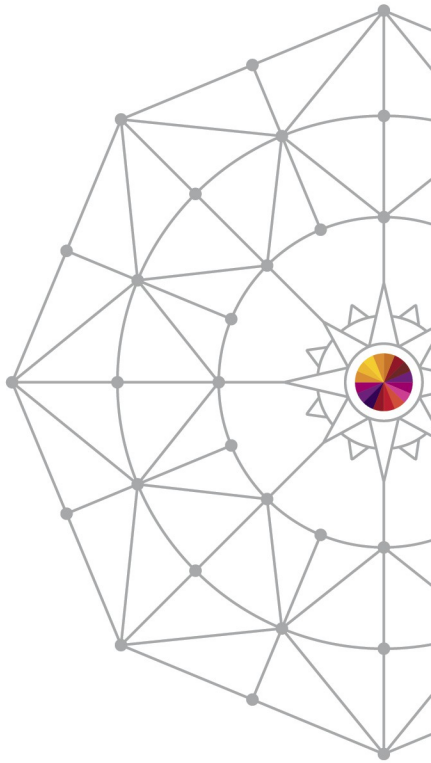
What's New in … Enterprise PL/I 4.4 and z/OS V2R1 XL C/C++

Dickson Chau
IBM

August 6, 2014
Session 16092



#SHAREorg

Copyright (c) 2014 by SHARE Inc. Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by-nc-sa/3.0/

SHARE in Pittsburgh 2014

Agenda

New Features in …

- z/OS V2R1 XL C/C++
  - Language
    - C11 standard
    - C++11 standard
    - Metal C
  - Usability
  - Performance
    - OpenMP API v3.1
- Enterprise PL/I v4.4
  - Middleware Support
  - Usability
  - Performance

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

z/OS V2R1 XL C/C++
C11 standard

- C11 is a new version of the C programming language standard
- Was known as C1X before its ratification
- New LANGLVL sub-option EXTC1X
  - Enable all currently supported C11 features
- Features are phased in towards full compliance
- Currently support:
  - Anonymous structures
  - Complex type initialization
  - Generic selection
  - Static assertions
  - The _Noreturn function specifier

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C11 standard
Anonymous Structures

- An anonymous structure is a structure that does not have a tag or a name and that is a member of another structure or union
- All the members of the anonymous structure behave as if they were members of the parent structure
- Example:

```
struct s {
  struct {
    // This is an anonymous structure, because it has no tag, no name, and is a
   member of another structure.
    int a;
  };
  int b;
} s1;

int main(void) {
  s1.a = 1;
  s1.b = 2;
  ...
}
```

- Already supported by C++

C11 standard
Complex Type Initialization

- Can initialize C99 complex types with the CMPLX, CMPLXF, or CMPLXL macros
- The 2 values specified on the macro are of the form x + yi, where x and y can be any floating point value, including INFINITY or NAN.
- Example:

```
#include <stdio.h>
#include <math.h>
#include <complex.h>

int main(void) {
  float _Complex c1 = CMPLX(1.2, INFINITY);
  double _Complex c2 = CMPLXF(2.3, NAN);
  long double _Complex c3 = CMPLXL(4.5, 6.7);
  printf("c1 Value: %e + %e * I\n", __real__(c1), __imag__(c1));
  printf("c2 Value: %e + %e * I\n", __real__(c2), __imag__(c2));
  printf("c3 Value: %Le + %Le * I\n", __real__(c3), __imag__(c3));
  return 0;
}
```

```
> xlc -qlanglvl=extc1x -qfloat=ieee  a.c
>./a.out
c1 Value: 1.200000e+00 + INF * I
c2 Value: 2.300000e+00 + NaNQ(1) * I
c3 Value: 4.500000e+00 + 6.700000e+00 * I
```

- Supported as a C++ extension as well

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C11 standard
Generic Selection

- Provides a mechanism to choose an expression according to a given type name at compile time
  - A common usage of which is to define type generic macros
- Example:

```
#define myfunc(X) _Generic((X), \
long double:myfunc_ld, \
default:myfunc_d, \
float:myfunc_f \
)(X)
long double myfunc_ld(long double x) {printf("calling%s\n",__func__);}
double myfunc_d(double x) {printf("calling %s\n",__func__);}
float myfunc_f(float x) {printf("calling %s\n",__func__);}
int main(void)
{
  long double ld;
  double d;
  float f;

  ld=myfunc(ld);
  d=myfunc(d);
  f=myfunc(f);

  return 0;
}
```

```
> xlc -qlanglvl=extc1x a.c
> a.out
calling myfunc_ld
calling myfunc_d
calling myfunc_f
```

C11 standard
Static Assertions

- Can be declared to detect and diagnose common usage errors at compile time
  - _Static_assert
  - static_assert
    - defined in assert.h
- Example:

```
int main(void) {
  _Static_assert(sizeof(long)==4, "not using ILP32");
  return 0;
}

> xlc -qlanglvl=extc1x -q64 a.c
ERROR CCN3865 ./a.c:2     not using ILP32
```

- Already supported as a C++ extension

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C11 standard
The _Noreturn Function Specifier

- Declares a function that does not return to its caller
  - _Noreturn
  - noreturn
    - defined in stdnoreturn.h
- Function must call:
  - abort, exit, _Exit, longjmp, quick_exit, thrd_exit
- The compiler can better optimize your code without regard to what happens if it returns
- Example:

```
_Noreturn void quit() {
  exit(1);
}
```

- Supported as a C++ extension as well

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

z/OS V2R1 XL C/C++
C++11 standard

- Was known as C++0x before its ratification
- Continue to phase in new features toward full compliance
- Added in this release:
  - Explicit conversion operators
  - Generalized constant expressions
  - Scoped enumerations
  - Right angle brackets
  - Rvalue reference

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C++11 standard
Explicit Conversion Operators

- Can apply the explicit function specifier to the definition of a user-defined conversion function
  - Inhibit unintended implicit conversions through the user-defined conversion function
- Example:

```
#include <iostream>
template <class T> struct Ptr {
  Ptr():rawptr_(0) {}
  Ptr(T* Ptr):rawptr_(T) {}
  explicit operator bool() const {return rawptr_ != 0; }
  T* rawptr_;
};
int main() {
  int var1, var2;
  Ptr<int> ptr1, ptr2(&var2);
  ptr1 = &var1;
  if (ptr1)     //explicit bool operator provided - good.
    return 66;
  std::cout << "ptr1+ptr2= " << (ptr1+ptr2) << endl; //warning
  return 0;
}
```

```
> xlC -qlanglvl=explicitconversionoperators a.C
./a.C, line 14.40: CCN5218 (S) The call doesn't match any
parameter list for operator+.
./a.C, line 14.40: CCN6283 (I) builtin operator+(int, int) is
not a viable candidate.
```


Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C++11 standard
Generalized Constant Expressions

- Constant expression is an expression that can be evaluated at compile time
- A new constexpr specifier
- Example 1:

```
constexpr int i = 1;
```

- Example 2:

```
struct S {
  constexpr S(int i) : mem(i) {}
  private:
    int mem;
};

constexpr S s(55);
```

C++11 standard
Scoped Enumerations

- Example:

```
enum class color {red, white};
enum shape {square, circle};

int main() {
  color c1 = color::red;    // valid
  color c2 = white;         // invalid

  shape s1 = shape::square; // valid
  shape s2 = circle;        // valid

  …
}
```

C++11 standard
Right Angle Brackets

- The >> token can be treated as two consecutive > tokens
- Example:

```
const vector<int> vi = static_cast<vector<int>>(v);
```

- same as `const vector<int> vi = static_cast<vector<int> >(v);`
- 1st > token is treated as the ending delimiter for the template parameter list
- 2nd > token is treated as the ending delimiter for the static_cast operator

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

C++11 standard
Rvalue Reference

- An rvalue reference type is defined by placing the reference modifier && after the type specifier.
- Example:

```
string &string::operator=(string &&);
string a,b,c;
a = b + c;
```

This result in a move operator (AKA destructive copying) on the assignment.

If instead, using a normal copy assignment operator, which has this signature:

```
string &string::operator=(string &);
```

- requires a deep-copy of the temporary variable created from b+c
- discard the temporary variable

z/OS V2R1 XL C/C++
Metal C

- New enhancements to Metal C
  - Mixed addressing mode with IPA
    - Metal C applications with AMODE-switching requirements can take advantage of inter-procedural analysis optimization
    - Example:

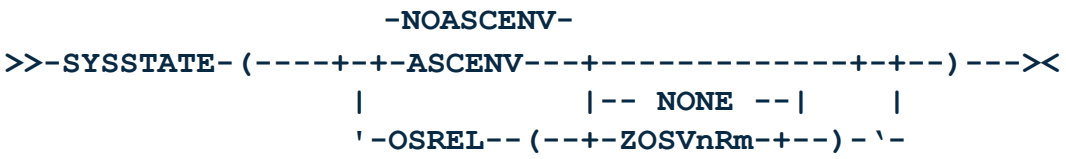
      xlc -qmetal -q32 -qipa -c 32.c

      xlc -qmetal -q64 -qipa -c 64.c

      xlc -qmetal -qipa -S 32.o 64.o
  - SYSSTATE option
  - User nominated main function

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Metal C
SYSSTATE Option



Default is SYSSTATE(NOASCENV,OSREL(NONE))

- OSREL=NONE|ZOSVnRm provides the value for the OSREL parameter on the SYSSTATE macro generated by the compiler. The value provided must be in the form of ZOSVnRm as described in the "z/OS MVS Programming: Assembler Services Reference". The default is NONE with which no OSREL parameter will appear on the SYSSTATE macro.
- ASCENV | NOASCENV ASCENV indicates to the compiler to automatically generate additional SYSSTATE macros with the ASCENV parameter to reflect the ASC mode of the function. The default is NOASCENV with which no ASCENV parameter will appear on the SYSSTATE macro.

Metal C
SYSSTATE Option (cont'd)

```
int main(void){
  int lv=256, addr;

  asm("  GETMAIN RC,LV=(%1),Key=(%2),LOC=(%3,%4) \n"
      "  ST 1,%0\n"
        :"=m"(addr)
        :"r"(lv),"I"(6),"I"(31),"I"(31)
        :"r1","r15");

__asm(" FREEMAIN RU,LV=256,A=(1) \n");

  return 0;
}

> xlc -qMETAL -S -qSYSSTATE=ASCENV test.c
```

- `SYSSTATE ASCENV=P` appears before function entry point marker
- If function compiled w/ -qARMODE or has `__attribute__((armode))`
  - `SYSSTATE ASCENV=AR` appears before function entry point marker

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Metal C
User Nominated main Function

```
#pragma map(main, "ANEWMAIN")
void dosomething(char *);

int main(int argc, char *argv[]) {
  for (int i=1; i<argc; i++) {
    dosomething(argv[i]);
  }
  return 0;
}
```

- When a Metal C program is built with the RENT option, it needs a “main” function to anchor the Writable Static Area (WSA) creation process. However a Metal C program may not have a function called “main” as the entry point thus not having the opportunity to be built with the RENT option.
- The entry point name in the generated code will be ANEWMAIN.
- When you link your program, you'll need to tell the binder that the entry point name is ANEWMAIN, such as this:

```
/bin/ld -o a.out a.o -e ANEWMAIN
```

z/OS V2R1 XL C/C++ Usability Enhancements

- include master header
  - INCLUDE option
  - To add additional include file(s) through option
- New C++ name mangeling sub-option
  - NAMEMANGLING(ZOSV2R1_ANSI)
- Debug improvement
  - Debugging optimized code
  - Debugging inlined procedures

Usability Enhancements
Debugging Optimized Code

- It is hard to debug optimized code because:
  - The debugger doesn't know where to find the value of a variable, i.e. it can be in a register, not in memory
  - The code generated ordering may not match the source code ordering

- The Debug Optimize Code feature:
  - Creates different levels of snapshots of objects at selected source locations
  - Makes the program state available to the debugging session at the selected source locations
  - When stopping at the snapshot points, the debugger should be able to retrieve the correct value of variables

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Usability Enhancements
Debugging Optimized Code (cont'd)

- The granularity of the snapshot points is controlled by the DEBUG(LEVEL) sub-option:
  - DEBUG(LEVEL(2)): No snapshot points inserted
  - DEBUG(LEVEL(5)): Snapshot points inserted before and after: if-endif, function, loop, and the first executable line of a function
  - DEBUG(LEVEL(8)): Snapshot points inserted at every executable statement
  - The line number table will only contain entries for the snapshot points
  - The debugger can only stop at snapshot points when doing source view debugging
  - Applies to DWARF format and OPT(2)

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Usability Enhancements
Debugging Inlined Procedures

- In V1R13, we added debug information for inline procedures
  - Set entry breakpoint for all inline instances of a procedure
  - No debug information is provided for the parameters and local variables of the inline instances
    - Debugger cannot show the value of these objects
- V2R1 provides debug information for parameters and local variables of each inline instance of a procedure
- The debugger is able to set and show the values of the parameters and locals of an inline instance

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Usability Enhancements Debugging Code Example

```
> cat a.c
#include <stdio.h>

int foo(int input) {
    int a;
    a = input * 2;
    printf("a = %d\n", a);
    return a;
}

int main(void){
   int i = foo(1); // inlined
   int j = i + 3;
   return j;
}
```

Debugging Code Example (cont'd) DEBUG(LEVEL)

```
> xlc -qdebug[=level=2] -O2 a.c
> dbx a.out
(dbx64) stop in main
[1] stop in 'int main()'        File
    SYSTEM:/home/share/inline/a.c, Line 6.
(dbx64) run
…
[1] stopped in foo at line 6 in file "a.c"  ($t2)
    6       printf("a = %d\n", a);
(dbx64) step
a = 2
stopped in foo at line 12 in file "a.c"  ($t2)
   12      int j = i + 3;
(dbx64) step
stopped in foo at line 14 in file "a.c"  ($t2)
   14   }
(dbx64) step
FDBX0414: Program exited with a return code of 5.
```

```
> xlc -qdebug=level=8 -O2 a.c
> dbx a.out
(dbx64) stop in main
[1] stop in 'int main()'        File
    SYSTEM:/home/share/inline/a.c, Line 3.
(dbx64) run
…
[1] stopped in foo at line 3 in file "a.c"  ($t2)
    3   int foo(int input) {
(dbx64) print input
1
(dbx64) s
stopped in foo at line 5 in file "a.c"  ($t2)
    5       a = input * 2;
(dbx64) s
stopped in foo at line 6 in file "a.c"  ($t2)
    6       printf("a = %d\n", a);
(dbx64) s
a = 2
stopped in foo at line 7 in file "a.c"  ($t2)
    7       return a;
(dbx64) s
stopped in main at line 13 in file "a.c"  ($t2)
   13      return j;
(dbx64) s
stopped in main at line 14 in file "a.c"  ($t2)
   14   }
(dbx64) s
FDBX0414: Program exited with a return code of 5.
```


Complete your session evaluations online at www.SHARE.org/PittsburghEval

z/OS V2R1 XL C/C++ Performance Enhancements

- New ARCH/TUNE(7) defaults
  - matches new ALS to z9 on z/OS V2R1
- [NO]THREADED option
- Better instruction selection
- Built-in functions
  - Decimal-Floating-Point (DFP) zoned conversion
  - Packed decimal
  - Transactional memory
- GNU C/C++ language extensions and compatibility
  - __builtin_expect
  - Attribute always_inline
- OpenMP API 3.1

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements NOTHREADED option

- For user to assert their application is single-threaded
  - Allows for more aggressive optimization and can potentially increase run-time performance
- Defaults is THREADED

- Example:

```
xlc -qnothreaded single-threaded.c
xlc [-qthreaded] multi-threaded.c
```


Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements
Better Instruction Selection

- Compiler can now make better use of the z/Architecture
  - Traditional 32-bit instructions are used for 4-byte operations eg. char, short and int
  - Eliminates all the unnecessary zero- and sign-extensions
  - Reduces code size and path length => improves run-time
  - Can make use of the High-Word Facility
- It used to be
  - The entire 64-bit GPRs are always populated
  - 64-bit operations are performed
    i.e. 64-bit version of instructions are used
- For:
  - LP32 with HGPR, and a function that make use of long long
  - LP64

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements
DFP Zoned Conversion Built-in Functions

- Available with the DFP and ARCH(10) options
- Converts from zoned type to DFP type:

```
_Decimal128  __cxzt(void* source, unsigned char length,
  const unsigned char mask);
_Decimal64 __cdzt(void* source, unsigned char length,
  const unsigned char mask);
```

- Converts from DFP type to zoned type:

```
int __czxt(_Decimal128 source, void* result, unsigned
  char length, const unsigned char mask);
int __czdt(_Decimal64 source, void* result, unsigned
  char length, const unsigned char mask);
```


Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements
Packed Decimal built-in functions

- 6 new built-in functions are available for C/C++ programs to access the hardware packed-decimal instructions
  - Compare Decimal – CP
  - Add Decimal – AP
  - Subtract Decimal – SP
  - Multiply Decimal – MP
  - Divide Decimal – DP
  - Shift and Round Decimal - SRP
- C++ and Metal C users now can directly utilize packed-decimal instructions
  - C++ does not have support for packed decimal intrinsic type
  - Decimal instructions are not normally generated

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements
Transactional Memory Built-in Functions

- For exploitation of the Transactional-Execution Facility
- Multi-threaded applications can benefit from processors' “opportunistic locking” of memory blocks. This could result in fast lock-free execution where there is no conflict.
- Following intrinsics are provided:
  - `long __TM_simple_begin(void);`
  - `long __TM_begin(void* const TM_buff);`
  - `long __TM_end(void);`
  - `void __TM_non_transactional_store(void* const addr, long long const value);`
  - `long __TM_nesting_depth(void* const TM_buff);`

Performance Enhancements
TM Built-in Functions (cont'd)

- Transaction failure diagnostic functions:
  - `long __TM_is_user_abort(void* const TM_buff);`
  - `long __TM_is_named_user_abort(void* const TM_buff, unsigned char* code);`
  - `long __TM_is_illegal(void* const TM_buff);`
  - `long __TM_is_footprint_exceeded(void* const TM_buff);`
  - `long __TM_is_nested_too_deep(void* const TM_buff);`
  - `long __TM_is_conflict(void* const TM_buff);`
  - `long __TM_is_failure_persistent(long const result);`
  - `long __TM_is_failure_address(void* const TM_buff);`
  - `long __TM_failure_code(void);`
  - `void __TM_abort_assist(unsigned int num_aborts);`

Performance Enhancements
TM Built-in Functions Example

- without TM BIFs:

```
pthread_mutex_lock(&mutex);
for (int i=1; i<NUMELEM; ++i)
  tmp[i] = MAX(global[i],tmp[i-1]);
pthread_mutex_unlock(&mutex);
```

- with TM BIFs:

```
if(__TM_simple_begin()) {
   // transaction failed
   // non-zero on TM_begin means failed
   pthread_mutex_lock(&mutex);
}

// start of transaction
for (i=1; i<NUMELEM; ++i)
    tmp[i] = MAX(global[i],tmp[i-1]);
// end of transaction

if(__TM_end()) {
  // not in a transaction state
  pthread_mutex_unlock(&mutex);
}
```

Performance Enhancements __builtin_expect

```
if(__builtin_expect(x, 0)) {
   error();
   ...
}
```

- Here, we expect that x will be equal 0, and we will not execute the statement for this branch very frequently
- Providing this additional information to the compiler allows it to be exploited for optimization

Performance Enhancements
Attribute always_inline

- `__attribute__ ((always_inline))`

- Greater control for the end users to instruct the compiler to inline a function
  - Available with OPT

Performance Enhancements
OpenMP API 3.1

- Industry-standard API designed to create portable C/C++ applications to exploit shared-memory parallelism
- Users can create or migrate parallel applications to take advantage of the multi-core design of modern processors
- Consists of a collection of compiler directives and library routines
- New SMP option to allow OpenMP parallelization directives to be recognized
  - Only supported in 64-bit
  - Executable must be run under USS
  - Thread-safe version of standard library must be used inside the parallel regions
  - Not supported with Metal C

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Performance Enhancements
OpenMP API 3.1 example

```
int bar(void) {
  #pragma omp parallel for
  for (int i = 0; i < N; i++) {
    // executed in parallel by a # of threads
    …
  }
}
```

Latest z/OS C/C++ Compiler Service Info

- V2R1 – released on April 2014
  - UI6362, UI6444
- V1R13 – released on March 2014
  - UI15229
- V1R12 – released on Oct. 2013
  - UK98192

- For latest PTFs info. refers to
  http://www-01.ibm.com/support/docview.wss?uid=swg21108506

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Agenda

New Features in …

- z/OS V2R1 XL C/C++
  - Language
    - C11 standard
    - C++11 standard
    - Metal C
  - Usability
  - Performance
    - OpenMP API v3.1
- Enterprise PL/I v4.4
  - Middleware Support
  - Usability
  - Performance

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Enterprise PL/I v4.4 Middleware Support

- Improved SQL support
  - More helpful messages from the SQL preprocessor
  - EMPTYDBRM SQL preprocessor option
    - For users that want an empty DBRM when the source contains no EXEC SQL
  - Structures with arrays supported
  - Nicer commenting out of SQL statements

Enterprise PL/I v4.4 Middleware Support (cont'd)

- Improved IMS support
  - Base 64 encoding and decoding functions
    - **`base64encode8( p, m, q, n )`**
    - **`base64decode8( p, m, q, n )`**
    - **`base64encode16( p, m, q, n )`**
    - **`base64decode16( p, m, q, n )`**
  - XML normalization and cleaning functions
    - **`WHITESPACEREPLACE`**
    - **`WHITESPACECOLLAPSE`**
    - **`XMLCLEAN`**
  - LOCATES attributes and associated functions for sparse arrays

Enterprise PL/I v4.4
Usability Enhancements

- WIDEPIC attribute
  - For UTF-16 PICTURE support
- INDEXR built-in function
  - Searches, from the right, for the first occurrence of one string within another
- DEFAULT statement expanded
- New 2nd argument to ALLOCATE n bytes from an AREA
- CANCEL THREAD statement
- Nicer handling of INCLUDEs
  - Beginning and end of the include are marked by comments in the listing

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Enterprise PL/I v4.4
Performance Enhancements

- Improved exploitation of zEC12 and zBC12 hardware
- Faster listing generation
  - The code in the backend to generate the pseudo-assembler listing has been significantly improved
- Recommending STATIC
  - User programs contain declares which are arrays or structures that are declared with INIT attributes and never changed
  - This will generate a vast amount of code that will be executed every time the containing procedure is called. This is terrible for both compile-time and run-time performance
  - A W-level message for any declares
    - With more than 100 INITIAL items
    - But with a storage class other than STATIC

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Enterprise PL/I v4.4
Performance Enhancements (cont'd)

- The compiler is itself compiled with ARCH(7)
  - Still supports generating code with the ARCH(6) option
  - Support for ARCH(6) is likely to end in the near future
- Inlining of FIXED to WIDECHAR
  - For "nice" FIXED, conversion to WIDECHAR will be inlined by converting to ASCII and then converting the ASCII to WIDECHAR
    - "nice" FIXED BIN is (UN)SIGNED REAL FIXED BIN(p,0)
    - "nice" FIXED DEC is REAL FIXED DEC(p,q) with 0 <= q and q <= p
- Improved code for UTF-8 and UTF-16
  - Plus, some BIFs may now be used in restricted expressions

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Enterprise PL/I v4.4
Performance Enhancements (cont'd)

- More DFP instructions exploitation in PICTURE conversions
  - v4.3 only able to handle source precision was 9 or less
  - Now, extended to support source precision is 18 or less
  - Lesson learned from the Zoned-to/from-DFP Facility:
    - A longer set of instructions may be faster than a shorter set
    - Reducing storage references helps performance
    - Eliminating packed decimal instructions can help performance
    - Using decimal-floating-point may improve your code's performance even in program's that have no floating-point data
    - The compiler knows when these new ARCH(10) instructions will help and will exploit them appropriately for you

PICTURE to DFP Example

```
process float(dfp);
pic2dfp: proc( ein, aus )
  options(nodescriptor);
  dcl ein(0:100_000) pic'(9)9' connected;
  dcl aus(0:hbound(ein)) float dec(16) connected;
  dcl jx fixed bin(31);
  do jx = lbound(ein) to hbound(ein);
    aus(jx) = ein(jx);
  end;
end;
```


Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

PICTURE to DFP Example (cont'd)

- Under ARCH(9), the heart of the loop consists of these 17 instructions:

```
0060 F248 D0F0 F000   PACK #pd580_1(5,r13,240),_shadow4(9,r15,0)
0066 C050 0000 0035   LARL r5,F'53'
006C D204 D0F8 D0F0   MVC #pd581_1(5,r13,248),#pd580_1(r13,240)
0072 41F0 F009        LA r15,#AMNESIA(,r15,9)
0076 D100 D0FC 500C   MVN #pd581_1(1,r13,252),+CONSTANT_AREA(r5,12)
007C D204 D0E0 D0F8   MVC _temp2(5,r13,224),#pd581_1(r13,248)
0082 F874 D100 2000   ZAP #pd586_1(8,r13,256),_shadow3(5,r2,0)
0088 D207 D0E8 D100   MVC _temp1(8,r13,232),#pd586_1(r13,256)
008E 5800 4000        L r0,_shadow2(,r4,0)
0092 5850 4004        L r5,_shadow2(,r4,4)
0096 EB00 0020 000D   SLLG r0,r0,32
009C 1605             OR r0,r5
009E B3F3 0000        CDSTR f0,r0
00A2 EB00 0020 000C   SRLG r0,r0,32
00A8 B914 0011        LGFR r1,r1
00AC B3F6 0001        IEDTR f0,f0,r1
00B0 6000 E000        STD f0,_shadow1(,r14,0)
```


Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

PICTURE to DFP Example (cont'd)

- While under ARCH(10), it consists of just these 8 instructions

```
0060 EB2F 0003 00DF    SLLK r2,r15,3
0066 B9FA 202F         ALRK r2,r15,r2
006A A7FA 0001         AHI r15,H'1'
006E B9FA 2023         ALRK r2,r3,r2
0072 ED08 2000 00AA    CDZT f0,#AddressShadow(9,r2,0),b'0000'
0078 B914 0000         LGFR r0,r0
007C B3F6 0000         IEDTR f0,f0,r0
0080 6001 E000         STD f0,_shadow1(r1,r14,0)
```

  - The loop runs more than 4 times faster

Latest Enterprise PL/I Compiler Service Info

- V4.R4 – released on May 2014
  - UI18188, UI19089, UI18064, UI16996
- V4.R3 – released on May 2014
  - UI17889, UI19088, UI18063, UI17541
- V4.R2 – released on March 2014
  - UI15877, UK72572, UK97586, UK95453
- V3.R9 – released on Nov. 2013
  - UI12685, UK61147, UK90880, UK63656, UK83050

- For latest PTFs info. refers to

  http://www-01.ibm.com/support/docview.wss?uid=swg21170820

Recap on …
New Features in z/OS V2R1 XL C/C++

- C11 standard
  - Anonymous structures
  - Complex type initialization
  - Generic selection
  - Static assertions
  - The _Noreturn function specifier
- C++11 standard
  - Explicit conversion operators
  - Generalized constant expressions
  - Scoped enumerations
  - Right angle brackets
  - Rvalue reference

Complete your session evaluations online at www.SHARE.org/Pittsburgh-Eval

Recap on… (cont'd) new features in z/OS V2R1 XL C/C++

- Metal C:
  - Mixed addressing mode with IPA
  - SYSSTATE option
  - User nominated main function
- Debug improvements
- Performance enhancements
  - New ARCH/TUNE(7) defaults
  - Built-in functions
    - Decimal-Floating-Point (DFP) zoned conversion
    - Packed decimal
    - Transactional memory
- OpenMP API 3.1

References

- Peter Elderon, Enterprise PL/I 4.4 Highlights, SHARE Anaheim, Mar. 2014
- Visda Vokhshoori, What's New in z/OS XL C/C++ V2R1 Enterprise PL/I 4.4, SHARE Boston, Aug. 2013

Quick Survey

- Users of:
  - PL/I
  - C/C++
  - NOOPTIMIZE/OPTIMIZE(0), OPTIMIZE(2), OPTIMIZE(3)
  - ARCH(7), ARCH(8), ARCH(9), ARCH(10)
  - C/C++ only:
    - TUNE
    - LP64
    - PDF
    - HOT
    - IPA

Please join me tomorrow for …

- **Make Your PL/I and C/C++ Code Fly With the Right Compiler Options**
  - Session 16091
  - Thursday, August 7, 2014: 3:00 PM-4:00 PM
  - Room 317 (David L. Lawrence Convention Center)

Questions?

- Connect with us
  - Email me
  - Rational Café - the compilers user community & forum
    - C/C++: http://ibm.com/rational/community/cpp
    - PL/I: http://ibm.com/rational/community/pli
  - RFE community – for feature requests
    - C/C++: http://www.ibm.com/developerworks/rfe/?PROD_ID=700
    - PL/I: http://www.ibm.com/developerworks/rfe/?PROD_ID=699
  - Product Information
    - C/C++: http://www-03.ibm.com/software/products/us/en/czos
    - PL/I: http://www-03.ibm.com/software/products/en/plicompfami

Thank You!