GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

*****

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

*****

UNCLASSIFIED: Distribution Statement A.
Approved for public release; distribution is unlimited.

March 2012
GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

W. Bylsma
Dynamics and Structures
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
Detroit Arsenal
ATTN: RDTA-RS/MS157
6501 East 11 Mile Road
Warren, Michigan 48397-5000

*****

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

*****

UNCLASSIFIED: Distribution Statement A. Approved for public release; distribution is unlimited.
GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

WESLEY BYLSMA

DYNAMICS AND STRUCTURES–US ARMY RDECOM/TARDEC
ATTN: RDTA-RS/MS157
6501 E 11 MILE RD
WARREN, MI 48397-5000

This report documents GNU GCC options for interoperability between C/C++ and FORTRAN. Compiling legacy software on the Linux platform necessitated the usage of GNU compilers and changes to several intrinsic functions. Several miscellaneous related activities are also included. Experimentation was required to achieve success. Included are examples demonstrating particular usage of each. The functions addressed are TIME(), CTIME(), ETIME(), ITIME(), IDATE() (FORTRAN), getenv(), strncpy() (C), and vector<> (C++).
1.0 INTRODUCTION

This report documents GNU GCC options for interoperability between C/C++ and FORTRAN. Compiling legacy software on the Linux platform necessitated the usage of GNU compilers and changes to several intrinsic functions. Several miscellaneous related activities are also included. Experimentation was required to achieve success. Included are examples demonstrating particular usage of each. The functions addressed are TIME(), CTIME(), ETIME(), ITIME(), IDATE() (FORTRAN), getenv(), strncpy() (C), and vector<> (C++).

All examples were run on SUSE Linux Enterprise Server 11 (x86_64) with

- LSB_VERSION="core-2.0-noarch:core-3.2-noarch:core-4.0-noarch:core-2.0-x86_64:core-3.2-x86_64:core-4.0-x86_64"
- SGI ProPack 7SP1 for Linux, Build 701r3.sles11-1005252113
- SGI Tempo Service Node 2.1, Build 701r3.sles11-1005252113
- VERSION = 11
- PATCHLEVEL = 1

and using the GNU utilities

- gcc (SUSE Linux) 4.3.4 [gcc-4_3-branch revision 152973]
- GNU Fortran (SUSE Linux) 4.3.4 [gcc-4_3-branch revision 152973]
- GNU ld (GNU Binutils; SUSE Linux Enterprise 11) 2.20.0.20100122-0.7.9
- GNU ar (GNU Binutils; SUSE Linux Enterprise 11) 2.20.0.20100122-0.7.9

See references [1], [2], and [3].

2.0 C/C++ CALLING FORTRAN

This section demonstrates how to call FORTRAN subroutines from C/C++. A good reference is [4] that highlights these necessary considerations:

- extern "C" directive is used for modules not written in C++
- Fortran symbol references must be lowercase in C++ calls (C++ is case sensitive, Fortran is not)
- Usually Fortran appends an underscore (_) at the end of definitions and references to external symbols (subroutines, functions, etc.)
- C++ passes all parameters by value (except arrays and structures). Fortran passes them by reference
2.1 CHECK FOR 32 BIT AND 64 BIT LIBRARIES

This example shows how to call FORTRAN from C and tests for compatibility between 32 bit and 64 bit objects.

```
echo "\n\n"
echo "1-both 32-bit"
rm test.1
gfortran -m32 -c u.f
gcc -m32 u.o u.c -o test.1 -lgfortran
test.1

echo "\n\n"
echo "2-.f 32-bit"
rm test.2
gfortran -m32 -c u.f
gcc u.o u.c -o test.2 -lgfortran
test.2

echo "\n\n"
echo "3-both 64 bit"
rm test.3
gfortran -c u.f
gcc u.o u.c -o test.3 -lgfortran
test.3
```

u.c

```
#include <stdio.h>
extern test_();
int main() {
    printf("u.c: before call to test_\n");
test_();
    printf("u.c: after call to test_\n");
}
```

u.f

```
SUBROUTINE test
    print *,'in u.f'
END
```

OUTPUT:

1-both 32-bit
u.c: before call to test_
in u.f
u.c: after call to test_

2-.f 32-bit
rm: cannot remove `test.2': No such file or directory
/usr/lib64/gcc/x86_64-suse-linux/4.3.4/..../..../x86_64-suse-linux/bin/ld: i386 architecture of input file `u.o' is incompatible with i386:x86-64 output
collect2: ld returned 1 exit status
test.2: Command not found.

3-both 64 bit
u.c: before call to test_
in u.f
u.c: after call to test_
2.2 FORTRAN TIME() AND CTIME() USAGE

This example shows how to call FORTRAN from C and the usage of the FORTRAN TIME() and CTIME() intrinsic functions (see [5] and [6]). The code "u.c" in section 2.1 is used.

```c
rm test.1
gfortran -c ut.f
gcc ut.o ../section2.1/u.c -o test.1 -lgfortran
test.1
```

```
SUBROUTINE test
    CHARACTER*30 TimeSTR
    CHARACTER*8 TimeSTR1
    CALL CTIME(TIME(),TimeSTR)
    print *, 'TimeSTR:['||TimeSTR||']'
    TimeSTR1 = TimeSTR(12:19)
    print *, 'TimeSTR1:['||TimeSTR1||']'
    print *, 'TIME():'||TIME()'
END
```

OUTPUT:

```
  u.c: before call to test
  TimeSTR:[Wed Mar 7 09:56:38 2012 ]
  TimeSTR1:[09:56:38]
  TIME(): 1331132198
  u.c: after call to test
```

2.3 COMBINING OBJECTS INTO ONE EXECUTABLE

This example shows how to call FORTRAN from C, compile each object separately, and then combine the objects into an executable. The code in section 2.2 is used.

```c
   gfortran -m32 -c ../section2.2/ut.f -o f1.o
gcc -m32 -c ../section2.2/u.c -o c1.o
gcc -m32 f1.o c1.o -o cftest.e -lgfortran
cftest.e
```

The output below should be same output as above except for the time.

OUTPUT:

```
  u.c: before call to test
  TimeSTR:[Wed Mar 7 10:04:12 2012 ]
  TimeSTR1:[10:04:12]
  TIME(): 1331132652
  u.c: after call to test
```
2.4 C GETENV USAGE

This example shows the usage of the C GETENV() function (see [7]) and string manipulation using the tcsh shell to set the environment variable.

```bash
#!/usr/bin/tcsh
gcc uenv.c -o uenv.e
echo "\n\n---setenv"
setenv TESTENV /path1/path2/file
uenv.e
echo "\n\n---unsetenv"
unsetenv TESTENV
uenv.e
```

uenv.c

```c
#include <stdio.h>
#include <stdlib.h> /* getenv */
#include <string.h> /* strcpy */

int main() {
    char *str = "DUMMYENV";
    char *str1;
    char str2[50];
    printf("main: begin\n");
    printf("str:%s\n",str);

    str1 = getenv("TESTENV");
    if (str1 == NULL) {
        printf("NULL\n");
    } else {
        printf("str1:%s\n",str1);
        printf("len1:%d\n",strlen(str1));
    }
    strcpy(str2,str);
    str2[strlen(str2)-2] = 'X';
    str = str2;
    printf("str:%s\n",str);
    printf("len:%d\n",strlen(str));
    printf("main: end\n");
}
```

OUTPUT:

```
---setenv
main: begin
str:DUMMYENV
str1:/path1/path2/file
len1:17
str:DUMMYEXV
len:8
main: end

---unsetenv
main: begin
str:DUMMYENV
NULL
str:DUMMYEXV
len:8
main: end
```
3.0 FORTRAN CALLING C/C++

This section demonstrates how to call C/C++ subroutines from FORTRAN. A good reference is [8] and [9] where consideration is given to C++ name mangling (a technique used to allow function overloading due to C++'s object orientated foundations). Also notice the appended underscore (_) as noted in section 2.0 for FORTRAN function calls.

3.1 FORTRAN ETIME() USAGE

This example shows how to call C from FORTRAN and the usage of the FORTRAN ETIME() intrinsic function (see [10]).

```
rm f1.o c1.o fctest.e
gfortran -m32 -c u1.f -o f1.o -lgfortran
gcc -m32 -c u1.c -o c1.o
echo "=====nm"
 echo "f1.o"
nm f1.o
echo "---"
 echo "c1.o"
nm c1.o
echo "-----"
gfortran -m32 f1.o c1.o -o fctest.e
fctest.e

#include <stdio.h>
int pp_(int *a) {
    int k;
    printf("pp: begin\n");
    k = *a;
    printf("k:%d\n",k);
    printf("a*:%d\n",*a);
    printf("pp: end\n");
}
```
This example shows how to call C++ from FORTRAN. The code "u1.f" above is used.
rm f1.o c1.o fcpttest.e
echo "---call C++"
gfortran -m32 -c ../section3.1/u1.f -o f1.o -lgfortran
gcc -m32 -c u1.cpp -o c1.o -lstdc++
echo "-----nm"
echo "f1.o"
nm f1.o
echo "---"
echo "c1.o"
nm c1.o
echo "-----"
gfortran -m32 f1.o c1.o -o fcpttest.e -lstdc++
fcpptest.e

u1.cpp

```cpp
#include <iostream>
using namespace std;
extern "C" {
    int pp_(int *a) {
        int k;
        cout << "pp: begin" << endl;
        k = *a;
        cout << "k:" << k << endl;
        cout << "a*:" << *a << endl;
        cout << "pp: end" << endl;
        return 0;
    }
}
```
3.2 FORTRAN ITIME() AND IDATE USAGE

This example shows how to call C++ from FORTRAN and the usage of the FORTRAN ITIME() and IDATE() intrinsic functions (see [11] and [12]). The values returned in each array are summarized below:

<table>
<thead>
<tr>
<th>ITIME</th>
<th>IDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. hour, 1-24</td>
<td>1. day, 1-31</td>
</tr>
<tr>
<td>2. minute 1-60</td>
<td>2. month, 1-12</td>
</tr>
<tr>
<td>3. second 1-60</td>
<td>3. year</td>
</tr>
</tbody>
</table>

```
rm utcpp.o udcpp.o ut.o ut.e
gcc -m32 -c ut.cpp -o utcpp.o -lstdc++
gcc -m32 -c ud.cpp -o udcpp.o -lstdc++
gfortran -m32 -c ut.f -o ut.o -l_gfortran
gfortran -m32 ut.o utcpp.o udcpp.o -o ut.e -lstdc++
```
PROGRAM main
INTEGER TARRAY(3)
print *, 'main: begin'
CALL ITIME(TARRAY)
CALL pt(tarray)
print *, 'ta1:', tarray(1)
print *, 'ta2:', tarray(2)
print *, 'ta3:', tarray(3)
CALL IDATE(TARRAY)
CALL pd(tarray)
print *, 'ta1:', tarray(1)
print *, 'ta2:', tarray(2)
print *, 'ta3:', tarray(3)
print *, 'main: end'
END
This example shows how to call C from FORTRAN and the usage of C string functions to manipulate FORTRAN strings.

```fortran
rm f1.o c1.o fstr.e
gcc -m32 -c ustr.c -o c1.o
gfortran -m32 -c ustr.f -o f1.o -lgfortran
gfortran -m32 f1.o c1.o -o fstr.e
fstr.e

#include <stdio.h>
#include <string.h> /*strlen*/
int pp_(char *vi, int *lvi, char* vo, int *lvo) {
    char tmpstr[250];
    char fmt[250];
    char tt[] = "/path1/path2/file";
    printf("pp: start\n");
    printf("lvi:%d\n",*lvi);
    printf("vi:[%s]\n",tmpstr);
    strncpy(tmpstr, vi, *lvi);
    tmpstr[*lvi] = '\0';
    printf("fmt:[%s]\n",fmt);
    printf("fmt:[%s]n",tmpstr);
    sprintf(fmt,"%%%dc\0",6);
    printf("format:%%%dc\0",6);
    printf("fmt:[%s]\n",fmt);
    sprintf(fmt,fmt,' ');
    printf("fmt:[%s]\n",tmpstr);
    printf("fmt:[%s]\n",fmt);
    printf("pp: end\n");
    return;
} main: begin
pt: begin
    hour:10
    minute:59
    second:20
pt: end
    ta1:  10
    ta2:  59
    ta3:  20
pd: begin
    day:12
    month:3
    year:2012
pd: end
    ta1:  12
    ta2:  3
    ta3:  2012
main: end
```

3.3 FORTRAN STRING MANIPULATION USING C
3.4 FORTRAN ARRAY MANIPULATION USING C++

This example shows how to call C++ from FORTRAN and the usage of the C++ vector<> standard template library (STL) to manipulate FORTRAN arrays (see [13]).
PROGRAM main
DOUBLE PRECISION arr(3,2),arr1(3,2,3)
INTEGER i,j,siz(4)
print *, 'main: begin'
power(3, 2, arr)
for (int i=1;i<=3;i++) {
    cout << i << 
for (int j=1;j<=2;j++) {
    cout << j << 
for (int k=1;k<=3;k++) {
    cout << k << 
print *, 'i='boost::math::gamma_dist(2, 3, 2, 0)
do i=1,siz(2)
    print *, 'i=',i, ',arr(i,1), ','arr(i,2)
do i=1,siz(2)
print *, i, i, ',arr(i,1), ',arr(i,2)
do i=1,siz(2)
print *, 'i=',i, ',arr(i,1), ',arr(i,2)
do i=1,siz(2)
do j=1,siz(3)
do i=1,siz(2)
print *, i, j, k, arr(i, j, k)
enddo
enddo
CALL pa(siz, arr)
do i=1,siz(2)
print *, 'i=',i, ',arr(i,1), ',arr(i,2)
do i=1,siz(2)
print *, 'i=',i, ',arr(i,1), ',arr(i,2)
do i=1,siz(2)
do j=1,siz(3)
do i=1,siz(2)
print *, i, j, k, arr(i, j, k)
enddo
enddo
CALL pa(siz, arr)
do k=1,siz(4)
do j=1,siz(3)
do i=1,siz(2)
print *, i, j, arr(i, j)
enddo
enddo
print *, 'main: end'
END
```
main: begin
  arr(5,2) = arr(i,j)
    j=1,2
    i= 1  -1.92681655437397497E+220  6.64588727654947730E-316
    i= 2  4.09088348805498754E-314  2.1884619818311329E-314
    i= 3  4.94065645941246544E-324  -99.678901672363281
pa: begin
dim:2
  1 3
  2 2
sum:6
a.size:6
pa: end
  i= 1  1.00000000000000000  4.00000000000000000
  i= 2  2.00000000000000000  5.00000000000000000
  i= 3  3.00000000000000000  6.00000000000000000
arr1(3,2,3) = arr1(i,j,k)
i,j,k, val
  1 1 1  0.00000000000000000
  2 1 1 -1.92634262907847196E+220
  3 1 1  6.64588727654947730E-316
  1 2 1  0.00000000000000000
  2 2 1  0.00000000000000000
  3 2 1  0.00000000000000000
  1 1 2  0.00000000000000000
  2 1 2  0.00000000000000000
  3 1 2  0.00000000000000000
  1 2 2  0.00000000000000000
  2 2 2  0.00000000000000000
  3 2 2  0.00000000000000000
  1 1 3  0.00000000000000000
  2 1 3  0.00000000000000000
  3 1 3 -1.78035013165935472E+220
  1 2 3  NaN
  2 2 3 -1.92634255302512759E+220
  3 2 3 -99.678901672363281
pa: begin
dim:3
  1 3
  2 2
  3 3
sum:18
a.size:18
pa: end
  1  1  1  1.00000000000000000
  2  1  1  2.00000000000000000
  3  1  1  3.00000000000000000
  1  2  1  4.00000000000000000
  2  2  1  5.00000000000000000
  3  2  1  6.00000000000000000
  1  1  2  7.00000000000000000
  2  1  2  8.00000000000000000
  3  1  2  9.00000000000000000
  1  2  2 10.00000000000000000
  2  2  2 11.00000000000000000
  3  2  2 12.00000000000000000
  1  1  3 13.00000000000000000
  2  1  3 14.00000000000000000
  3  1  3 15.00000000000000000
  1  2  3 16.00000000000000000
  2  2  3 17.00000000000000000
  3  2  3 18.00000000000000000
main: end
```
4.0 MISCELLANEOUS

This section demonstrates several miscellaneous items associated with compiling C/C++ and FORTRAN.

4.1 PREPROCESSOR INCLUDE STATEMENTS

This example shows how to use include header files with FORTRAN.

```bash
gfortran -xf77-cpp-input ftest.f -o ftest
ftest
```

```c
#include "itest.h"

PROGRAM test
    print *, 'test: before a'
    print *, a = 12
    print *, a = 15
END
```

```
Integer a
print *, 'itest.h: set a to 15'
a = 12
a = 15
```

**OUTPUT:**

```
ittest.h: set a to 15
 test: before a
   a = 12
   a = 15
 test: after a
```

4.2 LARGE FILE SUPPORT (LSF)

This example shows how to include Large File Support (LSF) to read files greater than 2 GB on 32-bit systems (see [14], [15], and [16]). Programs have to be recompiled with "-D_FILE_OFFSET_BITS=64" which forces "off_t" and file access to 64 bits. An example with casting is below.

```c
(off_t) curpos = lseek(fd, (off_t)0, SEEK_CUR);
```

```bash
clear
echo "---32-bit"
gcc -m32 offsize.c -o csize
gcc -m32 offsize.cpp -o cppsize -lstdc+
csize
cppsize
echo ""
echo "---32-bit, w/-D_FILE_OFFSET_BITS=64"
gcc -m32 offsize.c -o cwsizze -D_FILE_OFFSET_BITS=64
gcc -m32 offsize.cpp -o cppwsizze -lstdc+ -D_FILE_OFFSET_BITS=64
cwsizze
cppwsizze
```
The output below shows the difference in size of “off_t” with the “-D_FILE_OFFSET_BITS=64” flag included—the C++ type “ios::off_type” is always 8 bytes in size.
4.3 CREATING AND COMPILING WITH ARCHIVES

This example shows how to create an archive and include it when compiling in the tcsh shell.

```bash
#!/usr/bin/tcsh
gcc -m32 -c s.c s1.c s2.c
echo "Creating Archive:"
ar rlusv libtest.a s1.o s2.o
ttrue libtest.a
echo " "
echo "Listing Archive:"
ar -tv libtest.a
nm libtest.a
echo " "
gcc -m32 s.o libtest.a -o ss.e
gcc -m32 s.o -o ss.e -L -liit.a
ss.e
```
s1.c

```c
#include <stdio.h>
void s1() {
    printf("in s1\n");
}
```

s2.c

```c
#include <stdio.h>
void s2() {
    printf("in s2\n");
}
```

s.c

```c
#include <stdio.h>
extern void s1();
extern void s2();
int main() {
    printf("Hello \n");
    s1();
    s2();
    printf("Hello \n");
}
```

```c
#include <stdio.h>
void s2() {
    printf("in s2\n");
}
```

```c
#include <stdio.h>
void s1() {
    printf("in s1\n");
}
```

OUTPUT:

Creating Archive:
ar: creating libtest.a
a - s1.o
a - s2.o

Listing Archive:
```
  rw-r----- 2048/14900  932 Mar 12 13:38 2012 s1.o
  rw-r----- 2048/14900  932 Mar 12 13:38 2012 s2.o
```

```
s1.o:
  U puts
  00000000 T s1

s2.o:
  U puts
  00000000 T s2

s: before s1,s2
  in s1
  in s2
s: after s1,s2
```
REFERENCES

[16] http://groups.google.com/group/comp.lang.c++.moderated/browse_thread/thread/ f4be9c112c05ef9b