Using C/C++ and Fortran together:

This tutorial covers mixing C/C++ and FORTRAN together, allowing C/C++ to call FORTRAN functions and FORTRAN to call C/C++ functions. Integrating C/C++ and FORTRAN into a single executable relies upon knowing how to interface the function calls, argument list and global data structures so the symbols match in the object code during linking. FORTRAN and C/C++ interoperability is sucessful when datatypes are matched appropriately, arguments passed correctly, the code linked properly and the executable runs without error.

Comparison of FORTRAN and C/C++ datatypes:

FORTRAN	C/C++
byte	unsigned char
integer*2	short int
integer	long int or int
integer iabc(2,3)	int iabc[3][2];
logical	long int or int
logical*1	bool (C++, One byte)
real	float
real*8	double
real*16	long double
complex	struct{float realnum; float imagnum;}
double complex	struct{double dr; double di;}
character*6 abc	char abc[6];
character*6 abc(4)	char abc[4][6];
parameter	#define *PARAMETER value*

This table shows the appropriate types to use for FORTRAN and C/C++ interoperability.

FORTRAN and C Arrays:

Order of multi dimensional arrays in C/C++ is the opposite of FORTRAN.

Native FORTRAN layout (collumn-major order): `INTEGER A(2,3)`

a(1,1)	a(2,1)	a(1,2)	a(2,2)	a(1,3)	a(2,3)

Or `INTEGER A(2,2,2)`

a(1,1,1)	a(2,1,1)	a(1,2,1)	a(2,2,1)	a(2,2,1)	a(1,1,2)	a(2,1,2	a(1,2,2)	a(2,2,2)

Native C layout (row-major order) is **NOT** equivalent to the FORTRAN layout: `int a[2][3];`

a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]

Thus:

- Equivalent multidimension arrays:
  - FORTRAN 77: `INTEGER I(2,3,4)`
  - FORTRAN 90: `INTEGER, DIMENSION(2,3,4) :: I`
  - C: `int i[4][3][2];`
- Accessing a FORTRAN array in C:
  Native FORTRAN:

```
   INTEGER   I,J,NI,NJ
   PARAMETER(NI=2,NJ=3)
   INTEGER   B(NI,NJ)                  same as B(2,3)

   DO J = 1, NJ                        loop to 3
       DO I = 1, NI                    loop to 2
           PRINT 10, I, J, B(I,J)
10         FORMAT(‘B(', I1, ',', I1, ') =', I2)
       END DO
   END DO
```

-

  Native C:

```
    int a[3][2])
    int i, j;

    for (i = 0; i < 3; i++)
        for (j = 0; j < 2; j++)
            printf(“a[%d][%d] = %d\n", i, j, a[i][j]);
```

- C array a[3][2] memory layout:

a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]

- In FORTRAN 90, also check out the "`RESHAPE`" directive.
- It is best not to re-dimension multi dimensional arrays within a function. Pass array size "n" and declare array as x[n][];

Linking FORTRAN And C Subroutines:

Fortran subroutines are the equivalent of "C" functions returning "`(void)`".

Note: The entry point names for some FORTRAN compilers have an underscore appended to the name. This is also true for common block/structure names as shown above.

FORTRAN	C
CALL SUBRA(...)	subra_(...)

The f77 comiler flags "`-fno-underscore`" and "`-fno-second-underscore`" will alter the default naming in the object code and thus affect linking. One may view the object file with the command `nm` (i.e.: `nm` *`file`*`.o`).

Note: The case in FORTRAN is NOT preserved and is represented in lower case in the object file. The g77 compiler option "`-fsource-case-lower`" is default. GNU g77 FORTRAN can be case sensitive with the compile option "`-fsource-case-preserve`".

NOTE: When debugging with GDB, the Fortran subroutines must be referenced with names as they appear in the symbol table. This is the same as the "C" representation. Thus when setting a break point at the Fortran subroutine subra(), issue the comand "`break subra_`".

Man pages:

- nm - list symbols from object files
- g77/f77

Function Arguments:

All arguments in FORTRAN are passed by reference and not by value. Thus C must pass FORTRAN arguments as a pointer.

FORTRAN	C

call subra(i, x)	subra_(int *i, float *x)

Character variables:

- Linux and GNU compilers: When passing character strings, the length must follow as separate arguments which are passed by value.

FORTRAN	C

call subra(string_a, string_b)	len_a = strlen(string_a);
	len_b = strlen(string_a);
	subra_(char *string_a, len_a, char *string_b, len_b)

- I have also seen the passing of a data structure containing two elements, the character string and an integer storing the length. This is common with databases such as Oracle.
- Classic AT&T UNIX:
  When passing character strings, the length must be appended as separate arguments which are passed by value.

FORTRAN	C
call subra(string_a, string_b)	subra_(char *string_a, char *string_b, len_a, len_b)

C expects strings to be null-terminated. Thus character strings from FORTRAN which are passed back to "C" should be null terminated with `CHAR(0)`

```
CHARACTER(LEN=32) :: sample_string = "This is a sample"//CHAR(0)

or

CALL SUBRA ('A string'//CHAR(0))
```

Alternate Returns:

FORTRAN	C

`call sub(a,b,c,*,*)`
`return 1`
`end`
	`int sub_(int *a,int *b,int *c)`
	`{`
	`    return(1)`
	`}`

`goto(1, 2, 3), sub()`	`if( sub() ) goto ERRS;`

Note: When using alternate returns to turn on/off an intlevel by returning a 1/0 you must use a FORTRAN wrapper to perform this function on the CSC Norwich mainframe. This is because the C compiler is old.

- **Buffering output:** Your machine may be configured where the output buffering defaults for FORTRAN and C may be configured the same or differently. C output may be buffered buffered while FORTRAN may not. If so, execute a function initialization task to unbuffer C otherwise print statements for C will be output out of order and at the end.

```
#include <stdio.h>

void ersetb(void){
    setbuf(stdout,NULL); /* set output to unbuffered */
}
```

Common Blocks:

Fortran common block and global C/C++ extern structs of same name are equivalent. Never use un-named common blocks! Reference variables in same order, same type and with the same name for both C and FORTRAN. Character data is aligned on word boundaries.

```
FORTRAN:
     DOUBLE PRECISION X
     INTEGER A, B, C
     COMMON/ABC/ X, A, B, C

C:
     extern struct{
         double x;
         int a, b, c;
     } abc_;

C++:
   extern "C" {
     extern struct{
         double x;
         int a, b, c;
     } abc_;
   }
```

Note: use of extern requires that the common block be referenced first by FORTRAN. If referenced first by C then drop the extern. The extern statement states that it is trying to reference memory which has already been set aside elsewhere.

[Potential Pitfall]: Byte alignment can be a source of data corruption if memory boundaries between FORTRAN and C/C++ are different. Each language may also align structure data differently. One must preserver the alignment of memory between the C/C++ "struct" and FORTRAN "common block" by ordering the variables in the exact same order and exactly matching the size of each variable. It is best to order the variables from the largest word size down to the smallest. Start with "double" followed by "float" and "int". Bool and byte aligned data should be listed last.

```
FORTRAN:
     INTEGER A, B, C
     DOUBLE PRECISION D, E, F
     LOGICAL*1  FLAG
     COMMON/ABC/ A, D, FLAG, B, E

C:
     extern struct{
         int a;
         double d;
         bool flag;
         int b;
         double e;
     } abc_;

C++:
   extern "C" {
     extern struct{
         int a;
         double d;
         bool flag;
         int b;
         double e;
     } abc_;
   }
```

Using GDB to examine alignment:

- Set a breakpoint in the C/C++ section of code which has visibility to the struct.
- While in a C/C++ section of code:

```
(gdb) print &abc_.b
$3 = (int *) 0x5013e8
```

- Set a breakpoint in the FORTRAN section of code which has visibility to the common block.
- While in a FORTRAN section of code:

```
(gdb) print &b
$2 = (PTR TO -> ( integer )) 0x5013e8
```

This will print the hex memory address of the variable as C/C++ and FORTRAN view the variable. The hex address should be the same for both. If not, the data will be passed improperly.

Forcing alignment with compiler arguments: Mixing Intel FORTRAN compiler and GNU g++: use the following compiler flags to force a common memory alignment and padding to achieve a common double word alignment of variables:

- Intel FORTRAN: `-warn alignments -align all -align rec8byte`

- Intel C/C++: `-Zp8`
- GNU g++: `-Wpadded -Wpacked -malign-double -mpreferred-stack-boundary=8`
  Example warning: `warning: padding struct size to alignment boundary`
- GNU g77: `-malign-double`

Example:

FORTRAN program calling a C function:

testF.f

```
      program test

      integer ii, jj, kk
      common/ijk/ ii, jj, kk
      real*8  ff
      character*32 cc

      ii = 2
      jj = 3
      kk = 4
      ff = 9.0567
      cc = 'Example of a character
string'

      write(6,10) ii, ff
10    format('ii= ',i2,' ff= ',f10.4)

      call abc(ii)

      write(6,20) ii
20    format('ii= ',i2)

      write(6,30) ii, jj, kk

      call doubleIJK(cc)

      write(6,30) ii, jj, kk
30    format('ii= ',i2,' jj= ', i2, ' kk=
', i2)

      write(6, 40) cc
40    format(a32)

      stop
      end

      subroutine abc(jj)
      jj = jj * 2
      return
      end
```

testC.c

```
#include <stdio.h>

extern struct
{
   int ii, jj, kk;
} ijk_;

int doubleijk_(char *cc, int ll)
{
   cc[ll--] = '\0';  // NULL terminate
the string

   printf("From doubleIJK: %s\n",cc);

   ijk_.ii *=2;
   ijk_.jj *=2;
   ijk_.kk *=2;

   return(1);
}
```

Compile:

- `f77 -c testF.f`
- `gcc -c testC.c`
- `f77 -o test testF.o testC.o`

Note: If there is use of C/C++ standard libraries you may have to include the following linking arguments: `-lc` or `-lstdc++`

Run: `./test`

```
ii=  2 ff=     9.0567
ii=  4
ii=  4 jj=  3 kk=  4
From doubleIJK: Example of a character string
ii=  8 jj=  6 kk=  8
Example of a character string
```

[Potential Pitfall]: If one does not terminate the string with NULL, the default with the new compilers (fort77 1.15 and gcc 4.5.2) and linker is to terminate the string with '\004' which is EOT (end of transmission). Previously this was not required but it is now. This code (`cc[ll--] = '\0';`) replaces the string terminating EOT character with a NULL.

C++ calling a FORTRAN function:

testC.cpp

```cpp
#include <iostream>

using namespace std;

extern"C" {
void fortfunc_(int *ii, float *ff);
}

main()
{

   int ii=5;
   float ff=5.5;

   fortfunc_(&ii, &ff);

   return 0;
}
```

testF.f

```fortran
      subroutine fortfunc(ii,ff)
      integer ii
      real*4  ff

      write(6,100) ii, ff
 100  format('ii=',i2,' ff=',f6.3)

      return
      end
```

Compile:

- `f77 -c testF.f`

- `g++ -c testC.cpp`
- `g++ -o test testF.o testC.o -lg2c`

Run: `./test`

```
      ii= 5 ff= 5.500
```

File I/O:

FORTRAN unit numbers can not be shared with "C" and "C" file pointers can not be used by FORTRAN. Pass data to a function (pick one language) which does the I/O for you. Use this function by both languages.

VAX Extensions:

The GNU FORTRAN compilers have only ported a small subset of VAX extensions. The vast majority will require a clever re-write.

VAX Variable Format Expressions:

VAX expression	Ported to GNU FORTRAN

`      integer*4 ivar(3), nfor`
`      nfor=3`
`      ...`
 	
`      write(6,100) (ivar(i), i=1,nfor)`
`  100 format(<nfor>I3)`
	`      do 20 i=1,nfor`
	`         write(6,200) ivar(i)`
	`  200    format(I3,$)  !! Supress carriage return`
	`   20 continue`
	
	`      write(6,400)     !! Write carriage return`
	`  400 format()`

VAX intrinsic functions:

Many are not supported in GNU FORTRAN and require the creation of an equivalent library written in "C".

Return type	VAX FORTRAN intrinsic function	Argument type

Integer*4	NINT(arg)
	JNINT(arg)
		Real*4

Integer*4	IDNINT(arg)
	JIDNINT(arg)
		Real*8

Integer*2	ININT(arg)	Real*4
Integer*8	KNINT(arg)	Real*4
Integer*2	IIDNNT(arg)	Real*8

Integer*8	KIDNNT(arg)	Real*8
Integer*2	IIQNNT(arg)	Real*16

Integer*4	IQNINT(arg)
	JIQNNT(arg)
		Real*16

Integer*8	KIQNNT(arg)	Real*16

Example: `inint.c`

```
short int inint_(float *rval)
{
   if(*rval < 0.0)
      return (*rval - 0.5);
   else
      return (*rval + 0.5);
}
```

- `gcc -c inint.c`
- `gcc -c idnint.c`
- ...
- Create library: `ar -cvq libvax.a inint.o idnint.o ...`

C++:

When mixing Fortran with C++, name mangling must be prevented.

```
          #ifdef __cplusplus
             extern"C" {
          #endif
          .
          .
          place declarations here
          .
          .
          #ifdef __cplusplus
          }
          #endif
```

For more on C++ name mangling, see the YoLinux.com C++ name mangling discussion.

The Intel FORTRAN compiler:

The Intel FORTRAN compiler has become a popular FORTRAN compiler for Linux as it supports many of the FORTRAN extensions supported by the Compaq (old DEC vax) and SGI FORTRAN compilers. (i.e. "structure", "record", "external", "encode", "decode", "find", "virtual", "pointer", "union", various intr8insic functions, I/O directives, ...) It links with object code generated by GNU gcc/g++ compilers as well as their own Intel C/C++ compilers.

Installation: (as root)

- `mkdir /opt/intel`
- `cd /opt/intel`
- move Intel Fortram compiler tar ball to this directory.
- `tar xzf 1_fc_c_9.0.033_ia32.tar.gz`
- `cd 1_fc_c_9.0.033/`
- `./install.sh`
  `1` : Install
  `2` : provide name of an existing license file.
  License file path :
  /path-to-license-file/`commercial_for_1_F2WC-5FDZV87R.lic`
  (file copied to `/opt/intel/licenses`)
  `1` (typical installation)
  Type "accept" to agree to license terms.
  Accept default location: `/opt/intel/fc/9.0`
  Accept default location: `/opt/intel/idb/9.0`
  `x` : Installation done

Compiler use and user configurations:

File: `$HOME/.bashrc`

```
..
...

#
# Intel Compiler
#

# FlexLM license server
export INTEL_LICENSE_FILE=28518@license-server

# Support for Intel FORTRAN compiler
if [ -f /opt/intel/fc/9.0/bin/ifortvars.sh ];
then
   source /opt/intel/fc/9.0/bin/ifortvars.sh
fi

# Support for Intel C/C++ compiler
if [ -f /opt/intel/cc/9.0/bin/iccvars.sh ];
then
   source /opt/intel/cc/9.0/bin/iccvars.sh
fi

# Support for Intel Debugger
if [ -f /opt/intel/cc/9.0/bin/idbvars.sh ];
then
   source /opt/intel/cc/9.0/bin/idbvars.sh
fi
```

```
...
...

export LD_LIBRARY_PATH
export PATH

```

File: `Makefile` (snipet)

```
F77=/opt/intel/fc/9.0/bin/ifort
F77FLAGS= -extend_source -fpp -f77rtl -intconstant -ftz -pad-source -sox \
          -lowercase -warn alignments -cxxlibgcc
CXX=g++
CPPFLAGS=-Wpadded -Wpacked
LDFLAGS=-L/opt/intel/fc/9.0/lib -lifport -lifcore -limf -Wabi -Wcast-align
DEBUG=-g

OBJS=file1.o file2.o

cpp-exe: $(OBJS)
       $(CC) $(LDFLAGS) -o name-of-exe $(OBJS)

.cpp.o:
     $(CXX) -c $(DEBUG) $(CPPFLAGS) $<

.f.o:
     $(F77) -c $(DEBUG) $(F77FLAGS) $<
```

Intel compiler directives:

Directive	Description
-extend_source	Length of line in source file allows for greater than 72 characters.
-vax	VAX FORTRAN runtime behavior. Changes read behavior. I never use this.
-f77rtl	FORTRAN 77 runtime behavior.
-fpp	Run preprocessor. i.e. handles `#define` and `#ifdef` macros.
-intconstant	Use FORTRAN 77 semantics to determine the kind of parameter for integer constants.
-ftz	Flushes denormal results to zero.
-pad-source	Specifies that fixed-form source records shorter than the statement field width should be padded with spacs (on the right) to the end of the field.
-lowercase	All function names are represented as lower case symbols.
-sox	Store compiler options and version in the executable.
-warn alignments	Warning if COMMON block records require padding for alignment.
-align all	Will get rid of warning: "`Because of COMMON, the alignment of object is inconsistent with its type [`*`variable-name`*`]` This requires a matching alignment for all code which links with this, C,

	C++ or FORTAN.

Use GNU g++ or Intel C++ compiler to compile and link with `main()`:

```
g++ -o name-of-exe main_prog.cpp file1.o file2.o -L/opt/intel/fc/9.0/lib -lifport -
lifcore -limf -Wabi -Wcast-align
```

Using `ddd` as a front-end for the Intel debugger:

```
      ddd --debugger "/opt/intel/idb/9.0/bin/idb -gdb" exe-file
```

[Potential Pitfall]: I found that I could not install the Intel FORTRAN compiler from an NFS mounted drive. I had to copy the Intel installation files to a local drive and install from there.

Documentation: /opt/intel/cc/9.0/doc/main_for/mergedProjects/bldaps_for/