G95 Fortran Lessons Learned

During our attempt to convert to Linux, it quickly became apparent that the included g77 compiler would not suit our needs. We had several pieces of code that used Fortran 90 components since we used the HP f90 compilers to compile. The effort to change the code to be f77 compliant would have been tremendous as well as provided ample opportunity to accidentally introduce bugs. Therefore, a search was made to find a compiler that would save us from this effort. G95 (www.g95.org) was the first one we stumbled upon. After downloading and compiling it, we attempted to compile our code with it. The results were a mixed bag but we slogged our way through and these are the lessons we learned from our journey.

HP -> X86 Linux Issues

It should be mentioned that there were a couple of issues that were not related to the fortran compilers but to the conversion from the HP platforms to the X86 Linux platforms.

1. **Endian Change.**
   One of the more headache-inducing differences between the two platforms is the change from Big Endian to Little Endian. Mcidas provides the swbyt4 routine to handle the conversion but it was difficult to know when you had to swap the bytes. Part of the problem lies in the fact that the some of the Mcidas routines handle the swap automatically, whereas some do not. Trial and error seemed the only way to figure this one out.
2. **Header Changes.**
   Some C header files were located in different directories so a few #includes had to be changed. The man pages were generally good enough to provide the appropriate path.
3. **Signal Code Changes.**
   The signal handler code for C had to be changed as well. For example, the sigvector call on the HP changed to sigaction. This change was not that difficult as the calls generally could be just swapped out. Again, the man pages were useful for understanding the change.

G77 -> G95 Syntax Errors

There were quite a few syntax changes to be made because of the stricter compiler as well as a few intrinsic name changes which might have been affected in g77 as well. The following list is in no particular order.

1. **Intrinsic DREAL -> REAL**
   This is a simple fix and is a carryover from the HP Intrinsics. The REAL function for HP f90 could return either a double or real based on the second

argument supplied to it. G95 has two separate functions to handle these: REAL and DREAL.

Ex. VAL1 = REAL(VAL2,4) -> VAL1=REAL(VAL2)
VAL1 = REAL(VAL2,8) -> VAL1 = DREAL(VAL2)

2. **Redeclaration Error**
   G95 by default does not like it when a variable is redeclared. If for some reason you need this ability then use the compiler flag: "-Wl,--allow-multiple-definition".
3. **Intrinsic STAT**
   For calls to the STAT intrinsic, the second argument's dimension had to be changed from 12 to 13 due to returning more information. We also had an issue with some programs that used STAT as a function name. These were changed to MYSTAT just to let it compile.
4. **Intrinsic IGETARG -> GETARG**
   The intrinsic IGETARG changed in two ways. One is the name: IGETARG is now just GETARG. The second change is that it only accepts two arguments now. IGETARG had a third argument for the number of characters to grab but that is no longer valid.
5. **DATA Z statements**
   DATA statements that initialize integer HEX values(or any Hollerinth) need to have have single quotes around the number.
   Ex. DATA MISS/Z80808080/ ->
   DATA MISS/Z'80808080'/
   For some reason, g95 does not allow characters to be initialized with HEX. Our solution was to move the assignment out of a DATA statement.
   Ex. DATA SPACE/Z20/ -> SPACE=Z'20'
6. **DO WHILE (1)**
   DO WHILE (1) is not valid in g95. DO is an equivalent statement that works.
7. **EQUIVALENCE after DATA statement**
   G95 is picky about EQUIVALENCE statements concerning variables that have already been initialized in a DATA statement. Simply move the EQUIVALENCE before the DATA statement to correct.
8. **CHAR in PARAMETER statement**
   Some of our code had PARAMATER statements that looked like:
   PARAMETER(CHNULL = CHAR(0))
   Our approach was to move the assignment down set it with a HEX value.
   Ex. CHNULL = Z'00'
   CHSPACE = Z'20'
9. **LOGICAL operators**
   The proper LOGICAL operators in g95 are .EQV. and .NEQV.
10. ***-1.0 Error**
    On some math expressions we were getting an error.
    Ex. Ssf=rsfparm(12,ihrf+1)*-1.0
    For some reason, the compiler could not handle the -1.0. To fix the problem we changed it to:
    Ssf = rsfparm(12,ihrf+1)*(-1.0)

We are thinking this is a bug but it is easy enough to work around.

11. **HP Alias**
    The HP compiler allowed you to use an alias reference a function. G95 does not have this feature. We just called the functions by their real name.
12. **Integer Pointer**
    G95 currently does not support an integer pointer. We had to recode a small program because of this. It was not a problem for us but could be a stumbling block if you are heavily dependent on this function.
13. **Compiler flags**
    In case you are curious these are the flags we are currently using for g95:
    -g –fno-second-underscore –Wno=101 –Wuninitialized –Wno-globals
    -fzero –fsloppy-char –ftrace=frame –Wl,--allow-multiple-definition

G77 -> G95 Runtime Errors

These are a list of the common runtime errors we have seen thus far.

1. **Fortran -> C strings**
   Since we have Fortran and C calling each other, we have found out that they don't always play nice. With g95 and gcc, this is more true than it used to be. When a Fortran is passed to C, it needs to be Null-terminated or C will have trouble and run off of the end. However, if you null-terminate the string that passed to the C function, don't expect any future compare on that string in Fortran to behave since the string is passed by reference from Fortran. We have found that as a general run, if you have to pass a string from Fortran to C, null-terminate a copy and send that instead.
2. **Data initialization**
   When we started running our programs, we noticed that some values seemed way off and some programs would bomb because of bad values. G95 by default does not initialize anything. DATA statements are the proper way, but the quick and dirty way is the compiler flag –fzero. This will initialize all *scalar* variables to 0. **NOTE:** The arrays still need a DATA statement.
3. **SAVE statements**
   If you have tried all of the fixes so far but something still is not working, a good test would be to start adding SAVE statements to your functions. Some of our code depended on the feature of the compiler doing a quiet SAVE for function variables. It seems g77 still does this. G95 does not and this is a troublesome problem to track down. If you function has ENTRY statements, be sure and include a SAVE in the top of the function.
4. **WRITE and STDOUT issue**
   We were trying to write some debug straight to stdout so that it would appear in the Mctext window but it would not show up. After much testing, it seems that g95 was not flushing stdout even after the process had finished. FLUSH(6) took care of the problem.

5. **Function return values**
   If for some reason you get a return value from a function that is an integer value of -2147483648, this a big clue that a function is not declared correctly. This usually happens when there is an IMPLICIT statement that covers the function but is the wrong data type. Just declare the function as the data type it should be and this should clear it up.