Hewlett Packard
Enterprise

Replacing common extensions in production codes with standard Fortran (and C)

Anton Shterenlikht and Harvey Richardson, HPE UK

Common extensions

- TIME
- ETIME
- Comma in READ/WRITE before input/output-item-list
- REAL array indices
- READ REAL into INTEGER
- Variable format expressions (VFE)
- Use association entity redeclaration
- Orphaned line continuation symbol, &
- Format specifiers without width
- Long lines
- .EQ. (==) for LOGICAL
- Mixing LOGICAL and INTEGER
- Function declaration with no ()
- + - with no bracket
- *kind-param* with no representation

All seen in production codes in 2020

Extensions - bad idea

- Reduces portability
  - Reduces opportunities for code testing with different compilers
  - Lower chances of finding coding error
  - Reduces opportunities for performance improvements
- Outdated, obsolete, unsafe code
- Code intention less clear – less code review – lower code quality
- Vendor lock

TIME

- GNU, Sun/Oracle, Intel, maybe others
- Simulates time(3) in C stdlib – seconds since 1-JAN-1970 00:00
- Production code – Tempo – pulsar timing data analysis: http://tempo.sourceforge.net

```
fmjdnow=40587+time()/86400.d0
                           ^
ftn-398 crayftn: ERROR TEMPO, File = src/tempo.f, Line = 261, Column = 29
  The generic interface "TIME" cannot be used as a function.
```

- Fix – declare time as `external` + `bind(C ... )`
- Module is better, but bigger change

TIME – standard conforming replacement

```
! C code

#include <time.h>
int c_time() { return (int) time(NULL); }



! Fortran code

integer function time()
  use, intrinsic :: iso_c_binding, only: c_int
  implicit none
  interface
    integer( c_int ) function c_time() bind(c, name='c_time')
      import c_int
    end function c_time
  end interface
  time = c_time()
end function time
```

– bind(C …) attribute

– Standard C

– Sun TIME returns 64-bin integer on 64-bit machines – use C_LONG

ETIME

- GNU, Sun/Oracle, IBM, Intel, maybe others
- Elapsed time from beginning of program execution
- Production code – EMAC – atmospheric chemistry: https://www.messy-interface.org

```
 87 #ifdef __ibm__
 88 #define etime etime_
 89 #endif
 90
 91 #ifndef __GFORTRAN__
 92     EXTERNAL  :: etime
 93 #endif
 94     INTRINSIC :: REAL
 95
 96     !  ---------------------------------------------
 97     !  returns elapsed cpu time since start of job (sec)
 98     !  ---------------------------------------------
 99     ! etime version; etime is common on many unix systems
100
101     ! I/O
102     REAL(DP), INTENT(OUT)  :: cpu,user,sys
103
104     ! LOCAL
105     REAL(SP), DIMENSION(2) :: tarray
106     REAL(SP)               :: etime
107
108     cpu  = REAL(etime(tarray), DP)
109     user = REAL(tarray(1), DP)
110     sys  = REAL(tarray(2), DP)
111
112   END SUBROUTINE cpu_second
```

– etime returns a real array

ETIME – standard conforming replacement

```c
#include <sys/times.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>

float c_utime() {
  struct tms tics;
  float utime;
  times( &tics );
  utime = tics.tms_utime *
   1.0/sysconf(_SC_CLK_TCK);
  return utime;
}

float c_stime() {
  struct tms tics;
  float stime;
  times( &tics );
  stime = tics.tms_stime *
   1.0/sysconf(_SC_CLK_TCK);
  return stime;
}
```

```
real function etime( tarray )
use, intrinsic :: iso_c_binding, only: c_float
implicit none
  interface
    real( c_float ) function c_utime() bind(c, name='c_utime')
      import c_float
    end function c_utime
    real( c_float ) function c_stime() bind(c, name='c_stime')
      import c_float
    end function c_stime
  end interface
  real, intent( out ) :: tarray(2)
  tarray(1) = c_utime()
  tarray(2) = c_stime()
  etime = tarray(1) + tarray(2)
end function etime
```

- Standard C but…
- POSIX only
- `sysconf`
- bind(C …)

If you actually care about timers

- Check they give the resolution you want and with low overhead if you need it

Some choices

- Fortran system_clock() (use selected_int_kind(18) args on Cray CCE).
- Programming model timers (MPI_Wtime() , omp_get_wtime())
- PAPI papi_get_real_cyc()
- gettimeofday(2)
- **clock_gettime(3)**

Comma in WRITE before *output-item-list*

- GNU, maybe others
- Origin and purpose unclear… perhaps to look like PRINT?
- Production code – Tempo – pulsar timing data analysis: http://tempo.sourceforge.net

```
          write (*,'(''ERROR: cannot use orbital frequency '',
     +          '' derivative fb'',i2,'' with binary model '',a8)'),
     +          i, bmodel(nbin)
```

Results in:

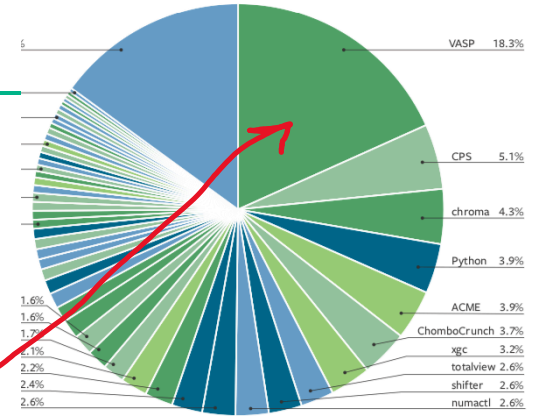
```
     +          '' derivative fb'',i2,'' with binary model '',a8)'),
                                                                 ^

ftn-1725 crayftn: ERROR RDPAR, File = inpar.f, Line = 1227, Column = 69
  Unexpected syntax while parsing the WRITE statement : "operand" was expected but found ",".
```

- Hard to automate – ”find comma after zero or more spaces after) matching the first (over multiple lines…”
- Fix – manual removal of comma

REAL array indices

- GNU, Intel, PGI (NVIDIA), maybe others
- Production code – VASP - ab-initio calculations - https://www.vasp.at



Top used code at NERSC: 18% of system usage[1]

```
2504
2505              CR(GRID%NINDPWCONJG(M))=CONJG(C(GRID%IND_IN_SPHERE(M)))*GRID%FFTSCA(GRID%IND_IN_SPHERE(M),2)
2506                                    ^
2507 ftn-319 crayftn: ERROR FFTWAV, File = fftw3d.f90, Line = 283, Column = 32
2508   A subscript must be a scalar integer expression.
2509                                                  ^
2510 ftn-319 crayftn: ERROR FFTWAV, File = fftw3d.f90, Line = 283, Column = 63
2511   A subscript must be a scalar integer expression.
2512                                                                        ^
2513 ftn-319 crayftn: ERROR FFTWAV, File = fftw3d.f90, Line = 283, Column = 99
2514   A subscript must be a scalar integer expression.
```

- Fix trivial in this case:

```
-    CR(GRID%NINDPWCONJG(M))=CONJG(C(GRID%IND_IN_SPHERE(M)))*GRID%FFTSCA(GRID%IND_IN_SPHERE(M),2)
+    CR(int(GRID%NINDPWCONJG(M)))=CONJG(C(int(GRID%IND_IN_SPHERE(M))))*GRID%FFTSCA(int(GRID%IND_IN_SPHERE(M)),2)
```

- But compiler documentation does not say whether they truncate (INT) or round (NINT)! So this extension is really "evil". Can cause all sorts of undetected bugs, e.g:
- User might assume that `element(1.999)` is `element(2)`, but it will likely be `element(1)`.
- And some compilers do not warn about this by default – beware!

[1] https://www.nersc.gov/assets/Uploads/2017NERSC-AnnualReport.pdf

READ REAL into INTEGER

- Intel, maybe others
- Production code – AFiD – CFD – turbulent flows – https://github.com/PhysicsofFluids/AFiD
- Compiler 1:

```
9709 lib-4190 : UNRECOVERABLE library error
9710   A numeric input field contains an invalid character.
```

- Compiler 2:

```
2855 At line 28 of file ReadInputFile.F90 (unit = 15, file = 'bou.in')
2856 Fortran runtime error: Bad integer for item 3 in list input
```

In `ReadInputFile.F90`:

```
17          integer   :: starea,tsta
28          read(15,*) flagstat,flagbal,tsta,starea
```

But in `bou.in`:

```
11  STATON  BALANCEON  TSTA  STAREAD
12  1        1           35.0   1
```

- Intel quietly converts `35.0` to `35`, but the code is broken! – the fix: `35.0 -> 35` in `bou.in`

Hewlett Packard Enterprise

Variable format expressions (VFE)

- Intel, Oracle, maybe others, but **not GNU**
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009
- Compiler 1:

```
                write(out,"(<sym%degen(igamma)>f18.8)") sym%irr(igamma,ioper)%repres(ideg,:)
                              ^

ftn-314 crayftn: ERROR CHECK_CHARACTERS_AND_REPRESENTATION, File = symmetry.f90, Line = 2568, Column
= 27

  Unknown edit descriptor "<" has been detected.
```

- Compiler 2:

```
 2568 |              write(out,"(<sym%degen(igamma)>f18.8)") sym%irr(igamma,ioper)%repres(ideg,:)
      |                            1

Error: Unexpected element '<' in format string at (1)
```

- "*Prerequisites: You will need Intel Compiler and Intel MKL*" – bad practice, avoid!

```
2569            do ideg = 1,sym%degen(igamma)
2570              write(out,"(<sym%degen(igamma)>f18.8)") sym%irr(igamma,ioper)%repres(ideg,:)
2571            enddo
```

Variable format expressions (VFE) - fix

- Use internal file:

```
--- symmetry.f90.orig 2020-09-07 09:23:23.000000000 -0500
+++ symmetry.f90 2020-09-07 09:37:36.000000000 -0500
@@ -2522,6 +2522,8 @@
     integer(ik) :: igamma,jgamma,ioper,ideg,iclass,ielem
     real(ark)   :: temp

+character(len=128) :: fmt
+
     do igamma = 1,sym%Nrepresen
       do jgamma = igamma,sym%Nrepresen
         !
@@ -2565,7 +2567,8 @@
           if (verbose_>=5) then
             write(out,"('igamma,iclass,ioper = ',3i6)") igamma,iclass,ioper
             do ideg = 1,sym%degen(igamma)
-              write(out,"(<sym%degen(igamma)>f18.8)") sym%irr(igamma,ioper)%repres(ideg,:)
+              write( fmt, * ) sym%degen(igamma)
+              write(out,"(" // ADJUSTL(fmt) // "f18.8)") sym%irr(igamma,ioper)%repres(ideg,:)
             enddo
           endif
```

Use association entity redeclaration

- Intel, PGI, GNU accept with no warning! - maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009
- CCE:

```
      integer(ik)           :: i,ik(1:3)
                                 ^

ftn-920 crayftn: ERROR MLDMS2PQR_XY2, File = pot_xy2.f90, Line = 2120, Column = 32
  "IK" is host associated, therefore it must not be redeclared with the INTEGER attribute.
```

- Actually the error message is wrong – "IK" is *use* associated, not host:

```
 1 module accuracy
...
10   integer, parameter :: ik          = selected_int_kind(8)


 5   use accuracy
```

- Violates 14.2.2 The USE statement and use association , para 9.
- Fix – rename!

Hewlett Packard Enterprise

Orphaned line continuation symbol, &

- Intel, GNU, maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009
- Compiler 1:

```
                                                                &
                                                                ^
ftn-703 crayftn: ERROR MLDMS2LOC_XY3, File = pot_xy3.f90, Line = 2831, Column = 76
  A continuation line can only follow a line continued with the "&" symbol.
```

- Compiler 2:

```
f951: Warning: '&' not allowed by itself in line 2831
```

- Violates 6.3.2.4 Free form statement continuation, para 1
- Fix – manual or a script

Format specifiers without width

- Intel, CCE, GNU (with `-fdec-format-defaults`), maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009
- GNU:

```
      176 |     if (verbose>=5) write(out,"('  size(work) = ',i)") iw
          |                                                       1
Error: Nonnegative width required in format string at (1)
```

- Violates 13.7.2.2 Integer editing, para 1
- Fix – add `0`! Change `i` into `i0`. "*When `w` is zero, the processor selects the field width*."

Long lines

- Intel, CCE, GNU, maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009

```
timer.f90:24:132:

   24 | blic TimerStart, TimerStop, TimerReport, TimerProbe, IOStart, IOStop , ArrayStart, ArrayStop, ArrayMinus, MemoryReport,memory_limit,maxmemory,
memory_now

      |                                                                                                                                    1

Error: Line truncated at (1) [-Werror=line-truncation]
```

- Violates 6.3.2.1 Free form line length: "*If a line consists entirely of characters of default kind (7.4.4), it shall contain at most 132 characters.*"
- Fix – manual or script, or…
- Most compilers have flags to deal with this, GNU: `-ffree-line-length-none`, or CCE: `-Ncol`

.EQ. (==) for LOGICAL

- Intel, maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009

```
              job%bset(i)%periodic   ==job%bset(i-1)%periodic   .and.job%bset(i)%iperiod    ==job%bset(i-1)%iperiod ) then
                                     ^
ftn-303 crayftn: ERROR FLREADINPUT, File = fields.f90, Line = 1860, Column = 40
  Data type LOGICAL is not allowed with LOGICAL for the operation "eq".
```

- Violates 10.1.5.1 Intrinsic operation classification.
- Fix – change `.EQ.` into `.EQV. :`

```
              job%bset(i)%periodic .EQV. job%bset(i-1)%periodic   .and.job%bset(i)%iperiod    ==job%bset(i-1)%iperiod ) then
```

Mixing LOGICAL and INTEGER

- Intel, GNU, maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009

```
                if (job%verbose>=5.and.mod(ientry,Nentries/50)) write(out,"('ientry = ',i)") ientry
                                       ^
ftn-303 crayftn: ERROR PREPARE_DIFF_EVIB, File = refinement.f90, Line = 3900, Column = 35
  Data type LOGICAL is not allowed with INTEGER for the operation "and".
```

- Intel and GNU interpret `.TRUE.` as `1`, `.FALSE.` as `0` and conversely `INTEGER` with lowest bit `0` as `.FALSE.` and with lowest bit `1` as `.TRUE.`
- 1,3,5,7… interpreted as `.TRUE.`
- *Probable* Fix (original intention not completely clear) – check that result of `MOD` is odd:

```
if (job%verbose>=5.and. (mod( mod(ientry,Nentries/50) ,2) .eq. 1) ) write(out,"('ientry = ',i)") ientry
```

Function declaration with no ()

- Intel, maybe others
- Production code – Theoretical ROVibrational Energies (TROVE) – variational nuclear motion solver - Fortran 2003 - https://doi.org/10.1016/j.jms.2007.07.009

```
   function FLread_extF_rank result (rank)
                             ^
ftn-1725 crayftn: ERROR FIELDS, File = fields.f90, Line = 17484, Column = 30
  Unexpected syntax while parsing the FUNCTION statement : "(" was expected but found "R".
```

- Violates R1530
- Fix – add `()` :

```
   function FLread_extF_rank() result (rank)
```

+ - with no bracket

- Intel, GNU, maybe others
- Production code – can’t say more…
- Compiler 1:

```
write (*,*) 1 + - 2
              ^

ftn-1725 crayftn: ERROR $MAIN, File = x.f90, Line = 1, Column = 17
  Unexpected syntax while parsing the WRITE statement : "operand" was expected but found "-".
```

- Compiler 2:

```
1 | write (*,*) 1 + - 2
|                 1

Warning: Extension: Unary operator following arithmetic operator (use parentheses) at (1)
```

- Fix obvious, but typically this code is a result of bad pre-processing, so better fix there

kind-param with no representation

- Production code – can't say more…

```
        1 integer, parameter :: r16 = kind(1._16)
PGF90-S-0081-Illegal selector - KIND parameter has unknown value for data type  (z.f90: 1)
  0 inform,   0 warnings,   1 severes, 0 fatal for MAIN
```

- 1) C715 (R708) "The value of *kind-param* shall specify a representation method that exists on the processor."
- 2) Different processors will used different numerical values for kind-param for the representation
- Portable code - use `SELECTED_REAL_KIND` or `REAL128` from `ISO_FORTRAN_ENV`

```
use, intrinsic :: iso_fortran_env, only : real128
if ( real128 .lt. 0) then
  write (*,*) "128-bit real kind not supported"
end if
if ( selected_real_kind(16,300) .lt. 0 ) then
  write (*,*) "real kind with PRECISION 16 digits and RANGE 300 not supported"
end if
end

$ ./a.out

 128-bit real kind not supported

 real kind with PRECISION 16 digits and RANGE 300 not supported
```

Hewlett Packard Enterprise

Other pain points if you deal with code from ‘others’

- Various and possibly uncommon build systems (fcm)
  Great when they work but not when they don’t.
- Meaning of extensions (.for, .F, .F90, …)
- Custom pre-processor phases in builds
- Code includes historic (not needed) or unsupported compiler directives:

```
183 !DIR$ IVDEP
184 !$DIR FORCE_VECTOR
185 !OCL NOVREC
```

  - Has anyone tried to standardise compiler code optimization directives?
- Variation in formatted output formats, in particular list-directed I/O (hard to compare correctness)
- Hard to parse variable names (using O2 l1)

Conclusions

- Avoid extensions
- Use maximum compiler diagnostic capabilities and act on all warnings
- Replace extensions with standard conforming portable code – not that hard, and well worth the effort.
- Check with multiple compilers